Introduction to Linear Regression

Prof. Nicholas Zabaras
Center for Informatics and Computational Science
https://cics.nd.edu/
University of Notre Dame
Notre Dame, Indiana, USA

Email: nzabaras@gmail.com
URL: https://www.zabaras.com/

January 28, 2019

Contents

- Chris Bishops' PRML book, Chapters 1 and 2
- Kevin Murphy's, Machine Learning: A probabilistic perspective, Chapter 5
- C P Robert, The Bayesian Choice: From Decision-Theoretic Motivations to Compulational Implementation, Springer-Verlag, NY, 2001 (online resource)
- A. Gelman, JB Carlin, HS Stern and DB Rubin, Bayesian Data Analysis, Chapman and Hall CRC Press, 2nd Edition, 2003.
- M Marin and C P Robert, The Bayesian Core, Spring Verlag, 2007 (online resource)
- Bayesian Statistics for Engineering, Online Course at Georgia Tech, B. Vidakovic.

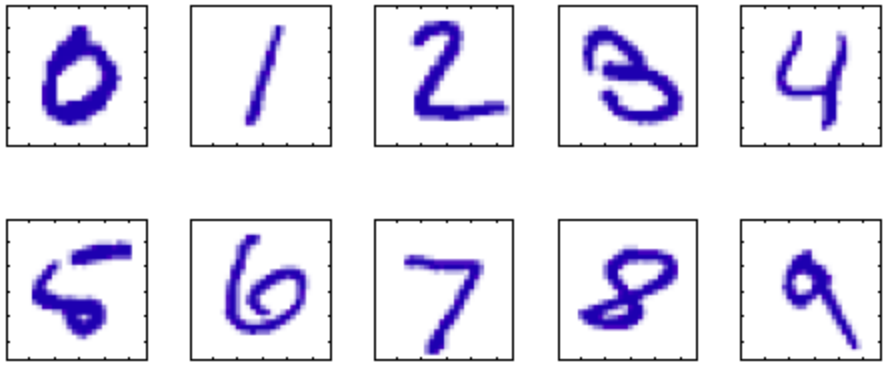
Typical Problems in Machine Learning

- Pattern Recognition: automatically *classifying the data* into different categories and use of these to take actions.

  - Example: handwritten recognition.

    Input: a vector $\boldsymbol{x}$ of pixel values.

    Output: A digit from 0 to 9.



Typical Problems in Machine Learning

- In the digits example, a large set of input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, or *a training set* is used to tune the parameters of an adaptive model.

- The category of an input vector is expressed using a target vector t (identifying the corresponding digit).

- The result of a machine learning algorithm: $y(\boldsymbol{x})$ where the output y is encoded as the target vectors for *any* input $\boldsymbol{x}$.

Terminology

- Training or learning phase: determine $y(\boldsymbol{x})$ on the basis of the training data.
- Test set, generalization
- Feature extraction
  - Data pre-processing (rotation, scaling, etc.)
  - Using lower-dimensional representation of the input and test data
- Supervised learning (input & target vectors in the training data)
  - Classification (discrete categories) or regression (continuous variables)

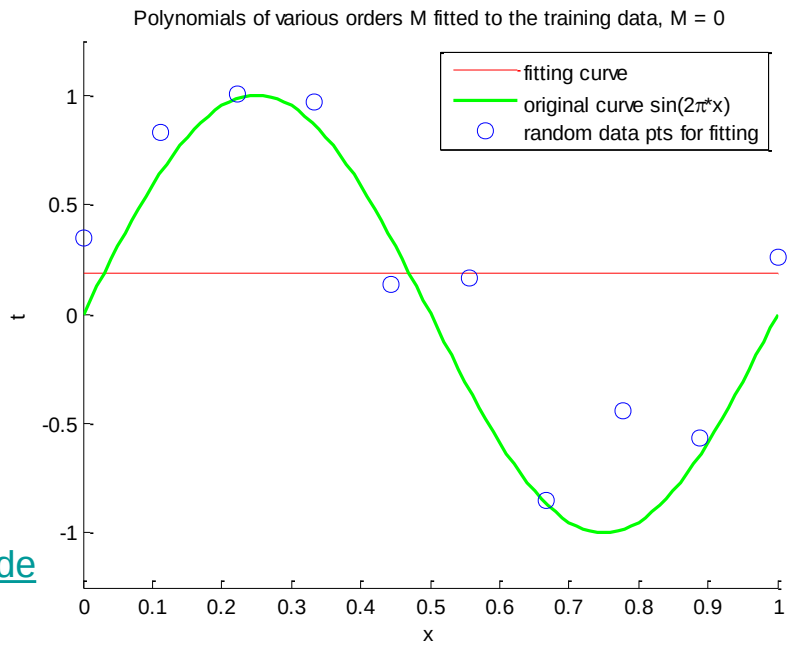
Terminology

- Unsupervised learning (no target vectors in the training data) also called clustering, or density estimation.
- Reinforcement learning (Richard S. Sutton and Andrew G. Barto)
  - credit assignment (rewards attributed to different moves at the end of a game)
  - exploration (of new actions)
  - exploitation (of high reward actions).

Motivation Example: Polynomial Curve Fitting

- Problem definition: implicitly trying to discover the underlying (generating) function $\sin(2\pi x)$ in a set of data.
- Some data points are known: $\boldsymbol{x} = (x_1, \ldots, x_N)^T$ as well as the corresponding target values $\boldsymbol{t} = (t_1, \ldots, t_N)^T$
- We fit the data using a polynomial function of the form

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + \ldots + w_M x^M = \sum_{i=0}^{M} w_i x^i$$



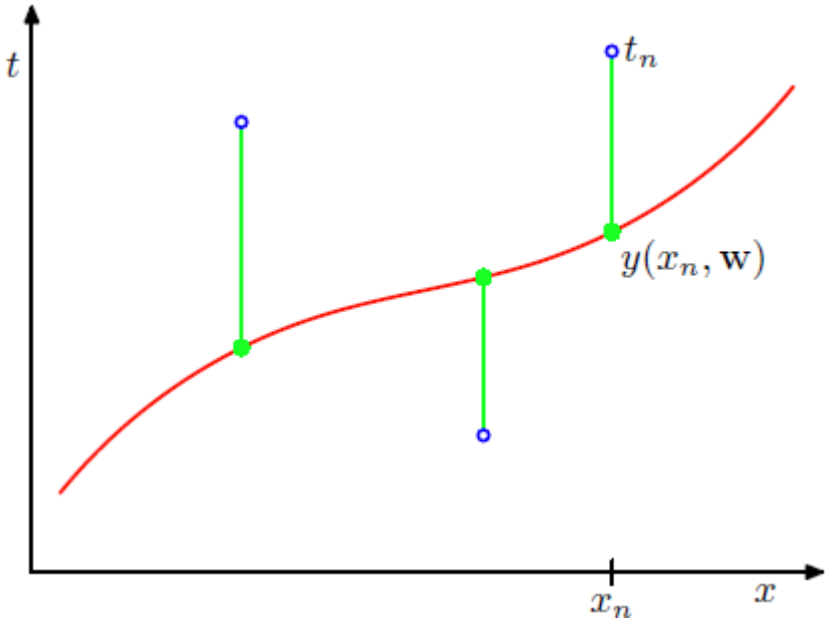
MatLab Code

Motivation Example: Polynomial Curve Fitting

- The values of the coefficients will be determined by fitting the polynomial $y(x, \boldsymbol{w}) = \sum_{j=0}^{M} w_j x^j$ to the training data.

- This can be done by minimizing an error function that measures the misfit between the function $y(x, \boldsymbol{w})$, for any given value of $\boldsymbol{w}$, and the training set data points.

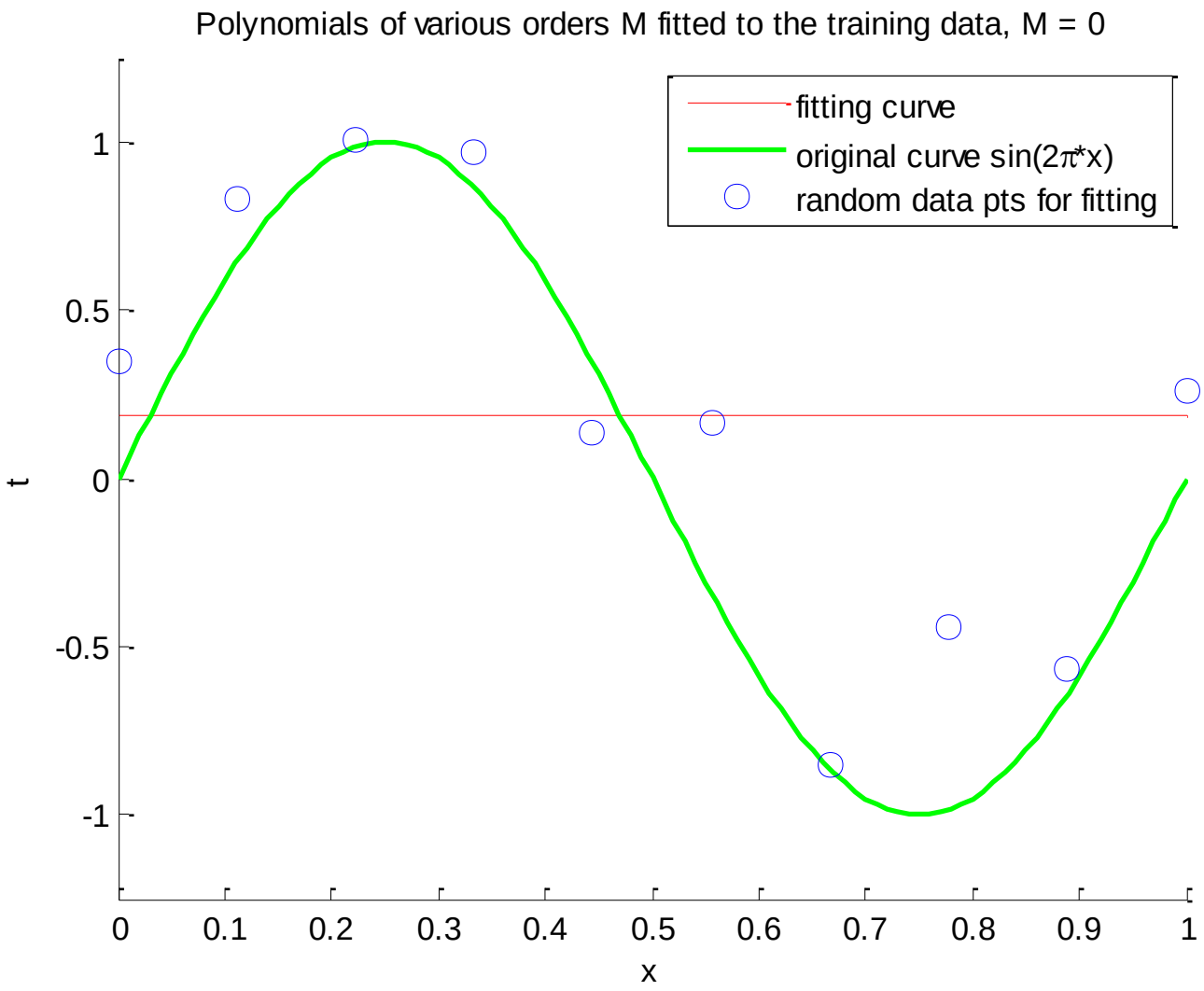
$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(y(x_n, \boldsymbol{w}) - t_n \right)^2$$



- You can show that the minimizer is obtained by solving:

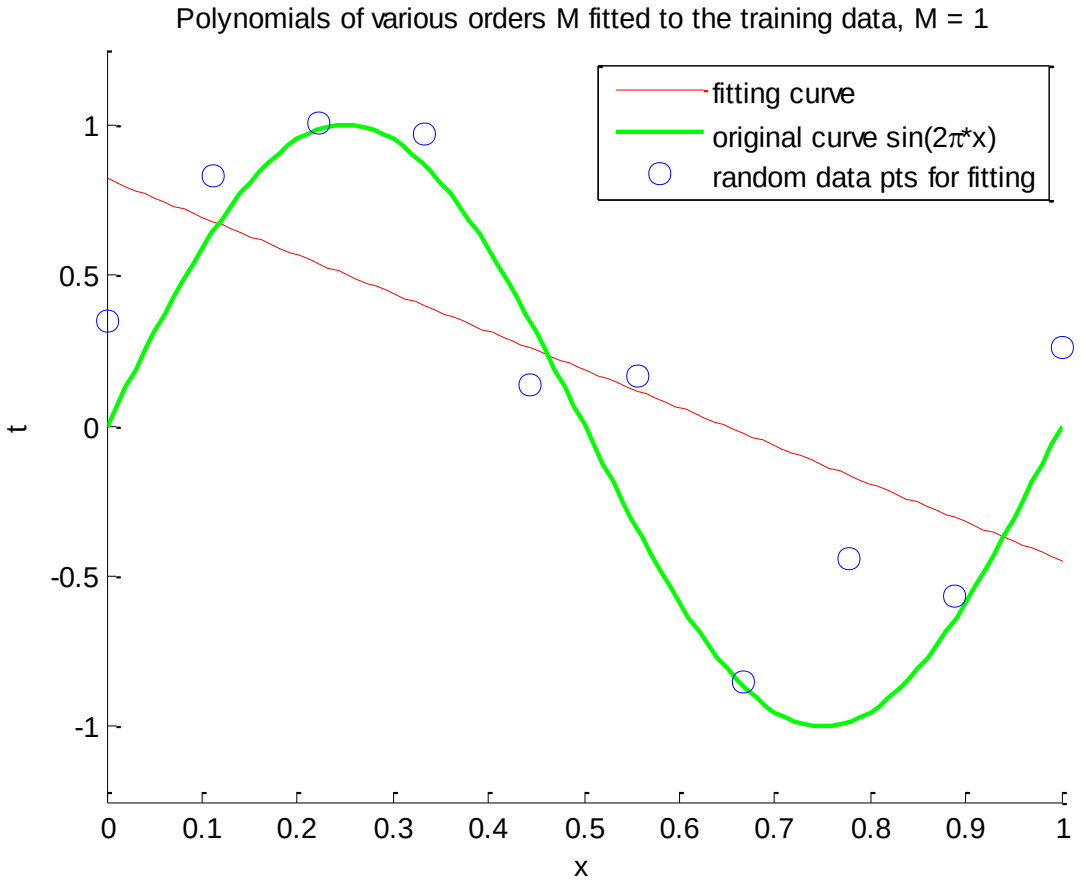
$$\sum_{j=0}^{M} A_{ij} w_j = T_i, \; A_{ij} = \sum_{n=1}^{N} x_n^{i+j}, \; T_i = \sum_{n=1}^{N} x_n^i t_n$$

Polynomial Curve Fitting



MatLab Code

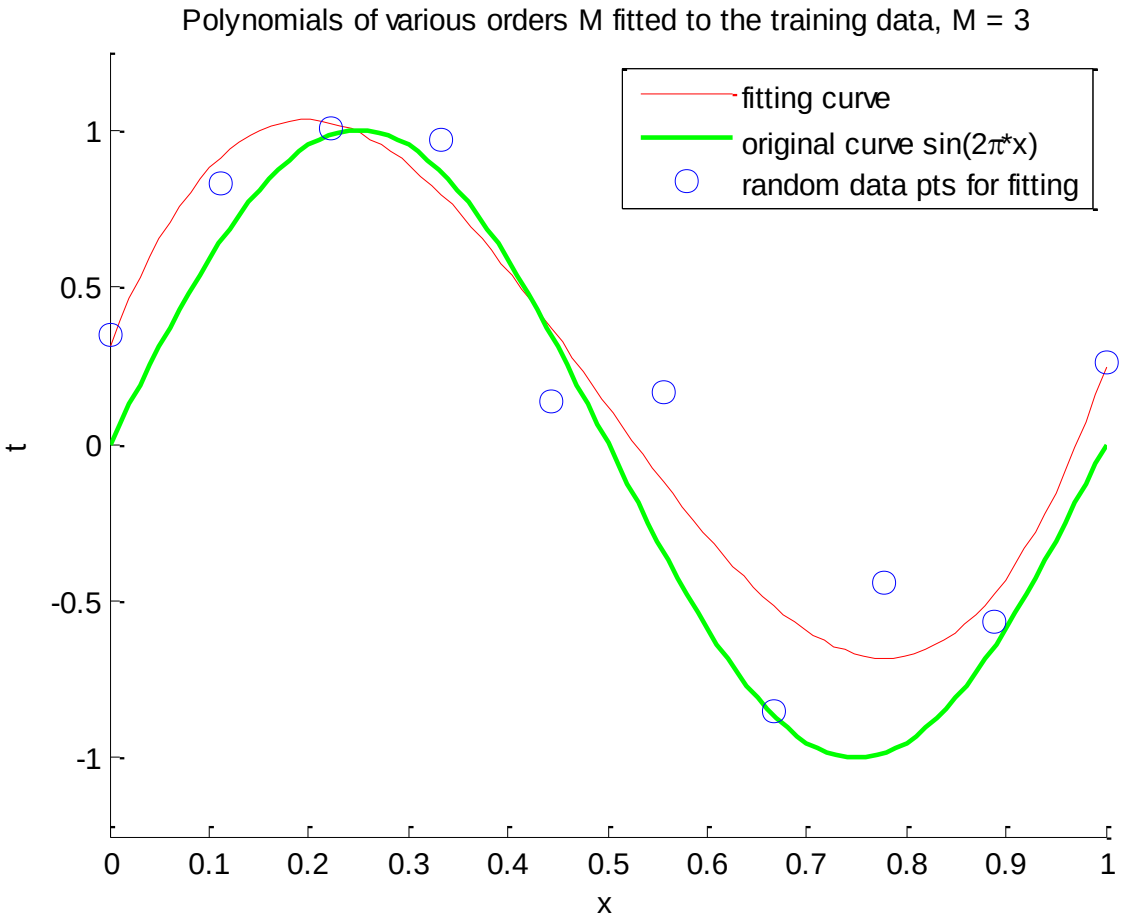
Polynomial Curve Fitting



MatLab Code

- 1st order (linear) polynomials give rather poor fit to the data and the $\sin(2\pi x)$.

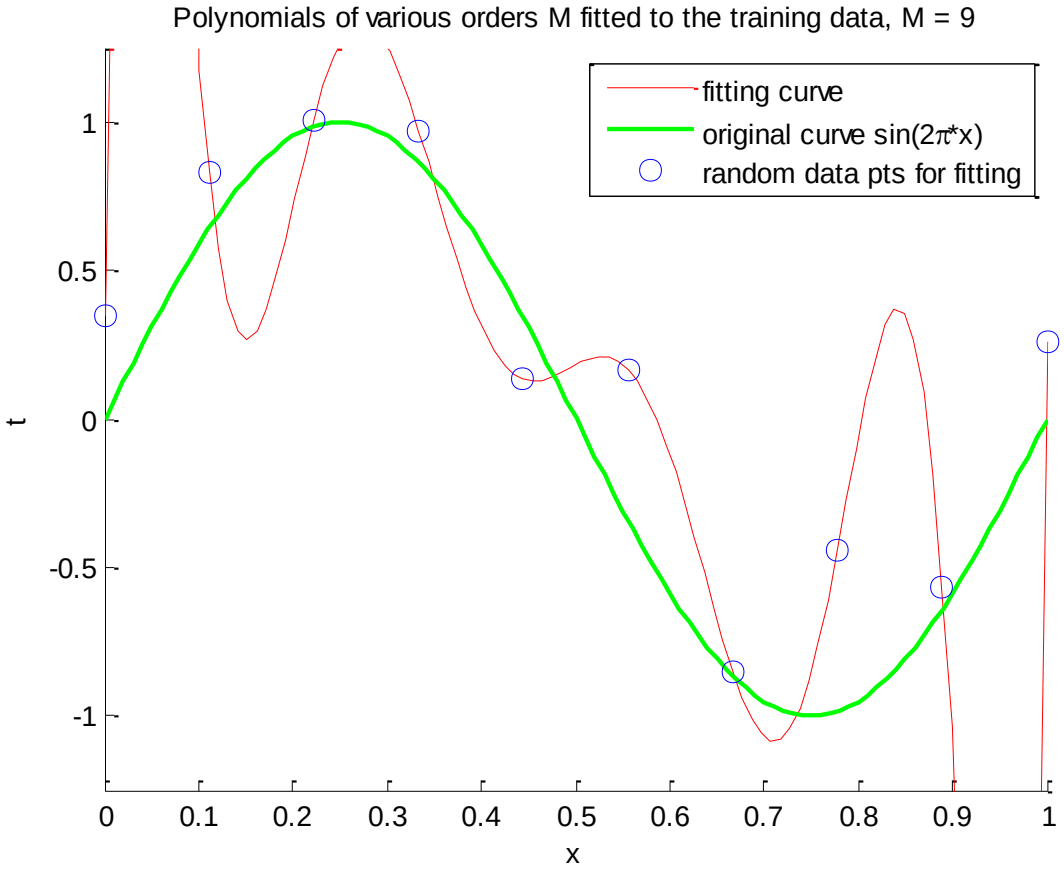
Polynomial Curve Fitting



MatLab Code

- The 3rd order polynomial seems to give the best fit to the function $\sin(2\pi x)$.

Overfitting



MatLab Code

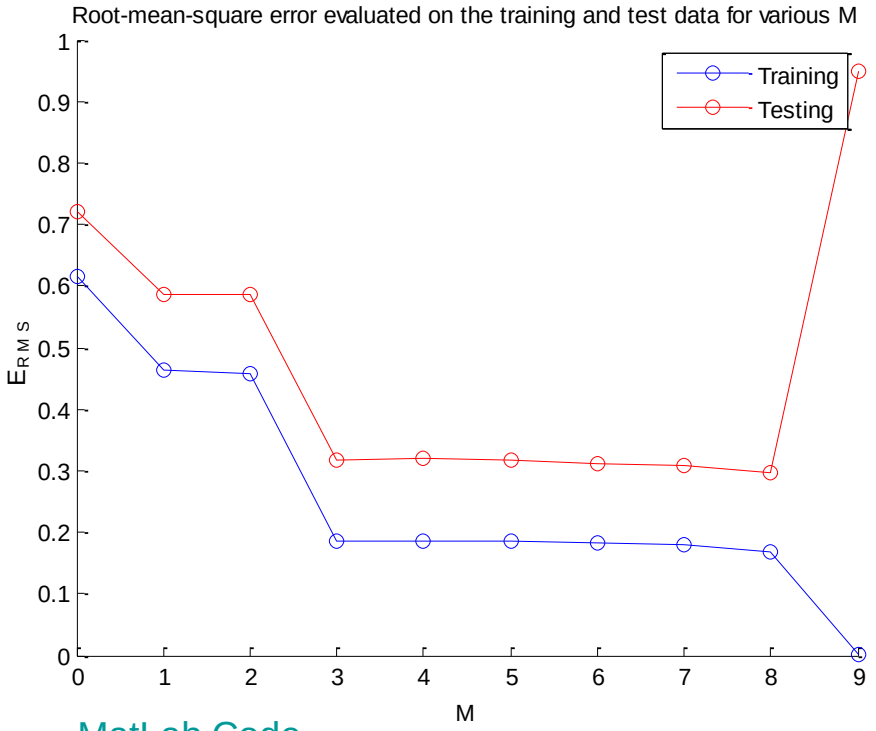
- For $M = 9$ we obtain a perfect fit to the training data. However, the fitted curve oscillates wildly and gives a very poor representation of $\sin(2\pi x)$. This is known as overfitting.

Training and Test Errors

- We often use the root-mean-square (RMS) error. *The division by N allows us to compare different data sizes. The square root makes E_{RMS} in units of t.*

$$E_{RMS} = \left(2E(\boldsymbol{w}^*)/N\right)^{1/2}$$

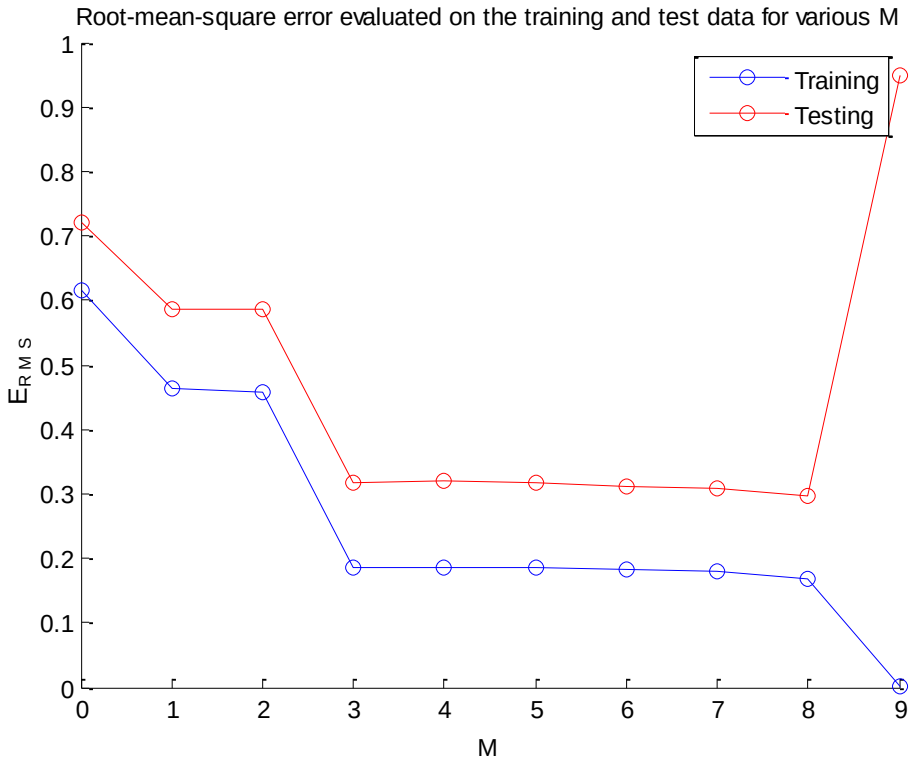
- The ***test set error*** is a measure of how well we are doing in predicting the values of t for new data observations of $\boldsymbol{x}$.



MatLab Code

Training and Test Errors

- Small values of M *give* relatively large values of the test set error. The corresponding polynomials are incapable of capturing the oscillations in $\sin(2\pi x)$.
- $3 < M \leq 8$ give small values for the test set error, and these also give reasonable representation of $\sin(2\pi x)$.



MatLab Code

Overfitting

- Obtain insight into the problem by examining the values of $\boldsymbol{w}$ obtained from polynomials of various order.
- As M increases, the magnitude of the coefficients typically gets larger.

1.0e+006 *

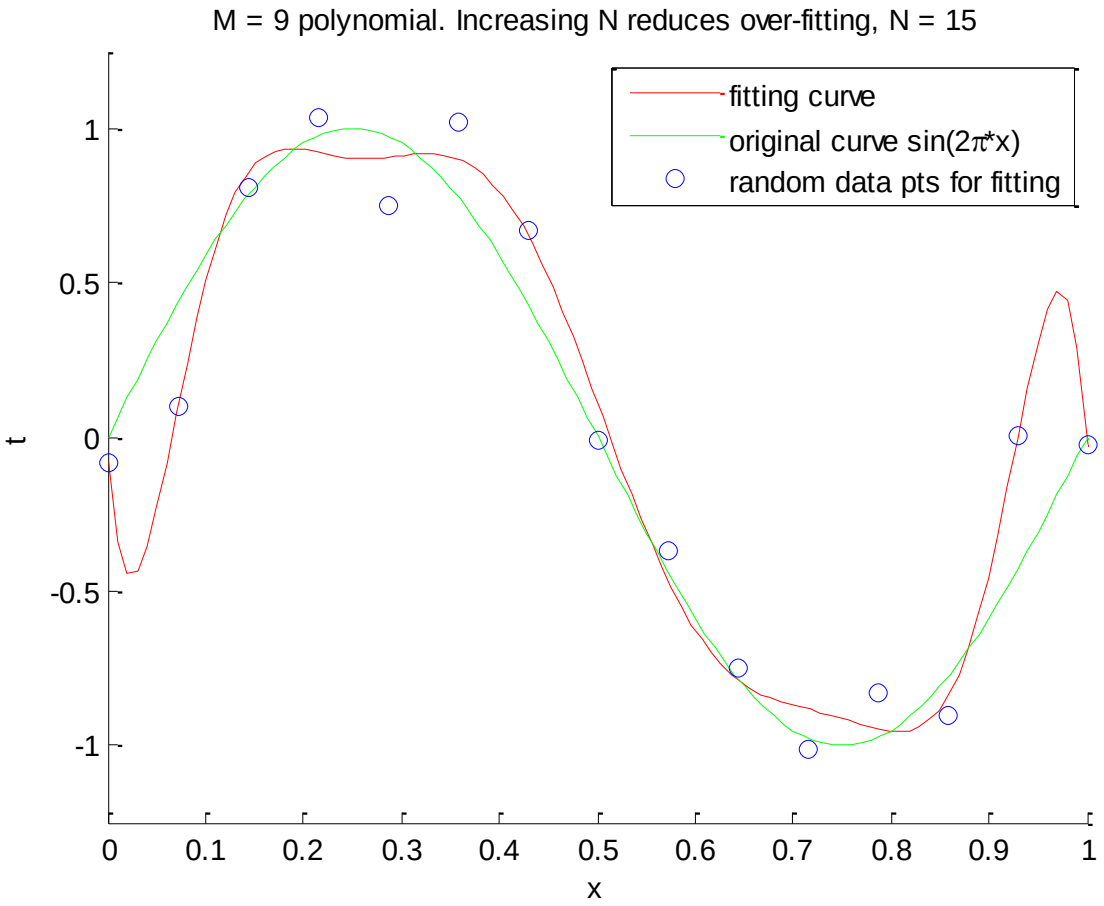
	$M=0$	$M=1$	$M=6$	$M=9$
w_0^*	0.0000	0.0000	0.0000	0.0000
w_1^*	0	-0.0000	0.0000	0.0002
w_2^*	0	0	-0.0000	-0.0053
w_3^*	0	0	0.0000	0.0486
w_4^*	0	0	0	-0.2316
w_5^*	0	0	0	0.6399
w_6^*	0	0	0	-1.0616
w_7^*	0	0	0	1.0422
w_8^*	0	0	0	-0.5576
w_9^*	0	0	0	0.1252

- For $M = 9$, the coefficients have become finely tuned to the data by developing large positive and negative values.
- The more flexible polynomials with larger values of M are becoming increasingly tuned to the random noise on the target values.

MatLab Code

Varying the Data Size

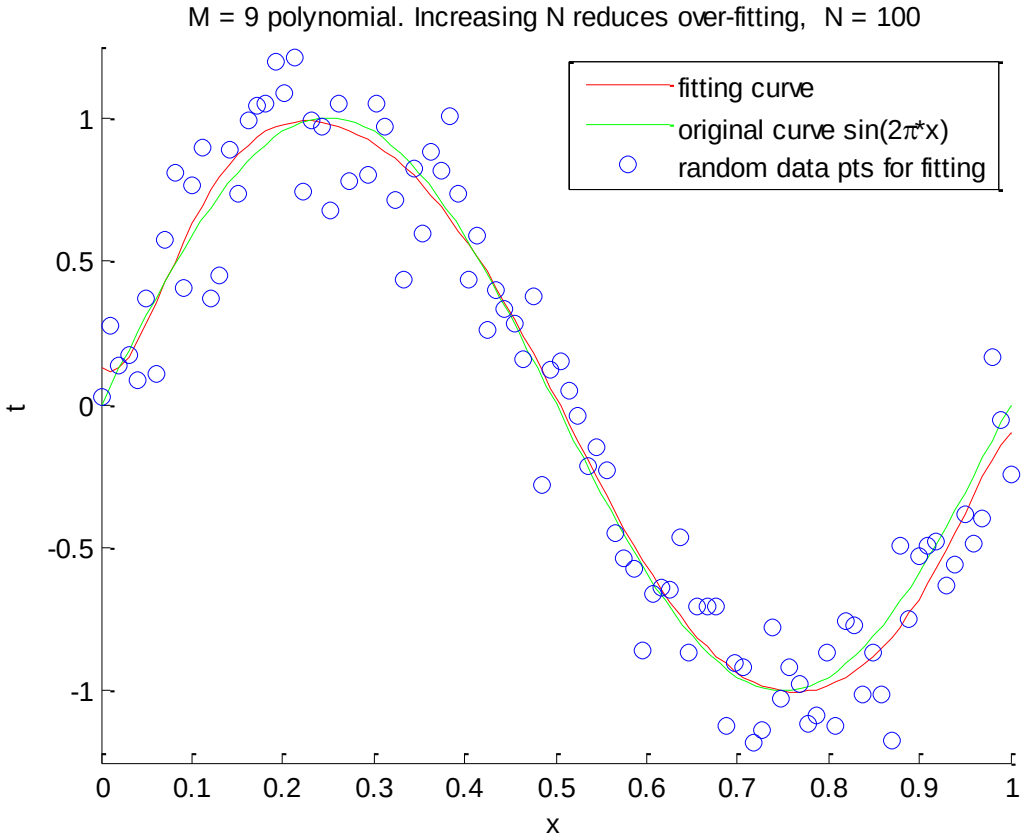
- It is also interesting to examine the behavior of a given model as the size of the data set is varied.



MatLab Code

Varying the Data Size

- For a given model complexity, the over-fitting problem becomes less severe as the size of the data set increases, i.e. the larger the data set, the more complex (in other words more flexible) the model that we can afford to fit to the data.



MatLab Code

Overfitting and Maximum Likelihood

- It is not reasonable having the number of parameters in a model (model complexity) limited according to the size of the available training set.

- It makes more sense to choose the complexity of the model according to the complexity of the problem being solved.

- The least squares approach to finding the model parameters is a specific case of maximum likelihood.

- The over-fitting problem is a general property of maximum likelihood.

Overfitting and Bayesian Approach

- By adopting a Bayesian approach, the over-fitting problem can be avoided.

- There is *no difficulty in employing models for which the number of parameters greatly exceeds the number of data points.*

- ***In a Bayesian model, "the effective number" of parameters adapts automatically to the size of the data set.***

Regularization Technique

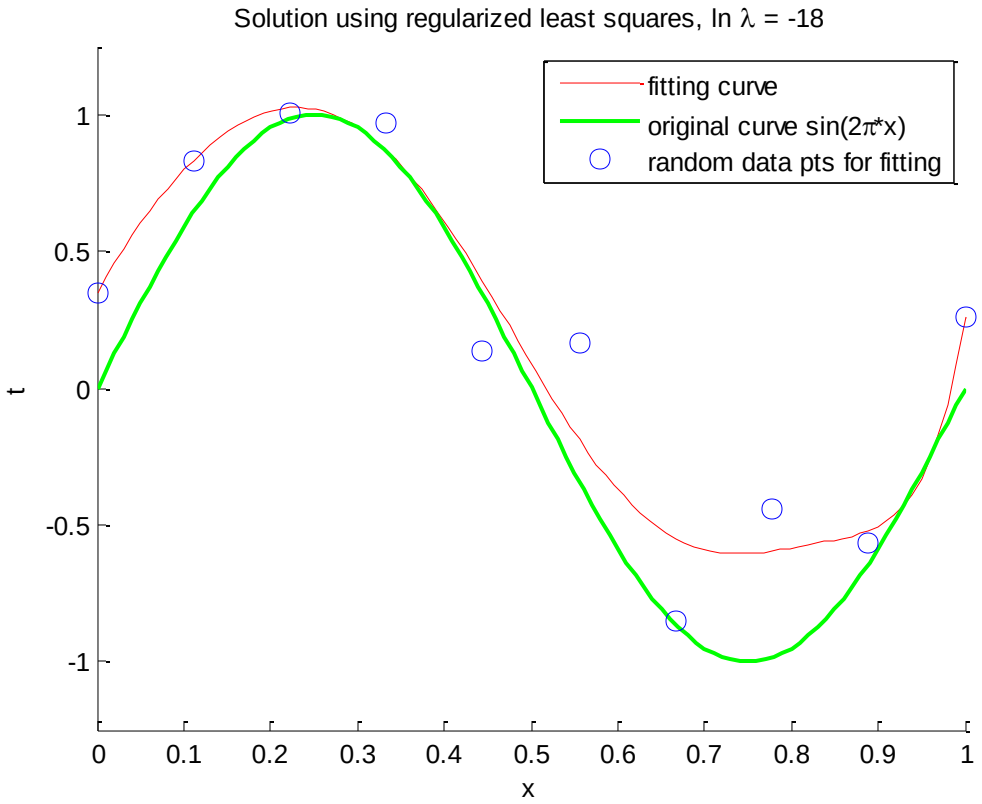
- To control the over-fitting we use *regularization, e.g.* adding "a penalty term" to the error function in order to discourage the coefficients from reaching large values.
- The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified error function of the form

$$\overline{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y(x_n,\boldsymbol{w}) - t_n\right)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

- The regularization parameter λ governs the relative importance of the regularization term compared with the sum-of-squares error term.
- The minimizer is similar to that given earlier but with

$$\sum_{j=0}^{M} A_{ij} w_j = T_i,\ A_{ij} = \sum_{n=1}^{N} x_n^{i+j} + \lambda\delta_{ij},\ T_i = \sum_{n=1}^{N} x_n^i t_n$$

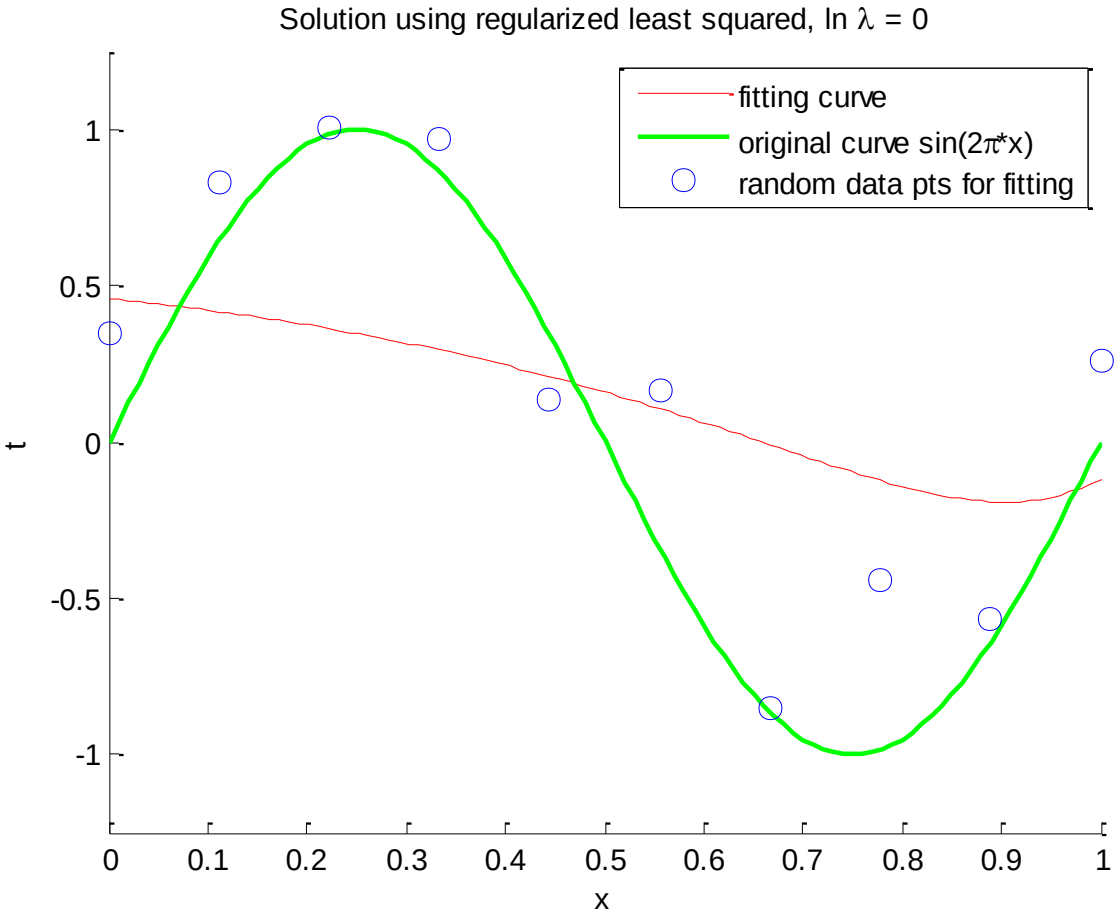
Regularization Controls Model Complexity



MatLab Code

- We now fit the polynomial of order $M = 9$ to the same data set as before but now using the regularized error function.
- For $lnl = -18$, the over-fitting has been suppressed and we obtain a much closer representation of $\sin(2\pi x)$.

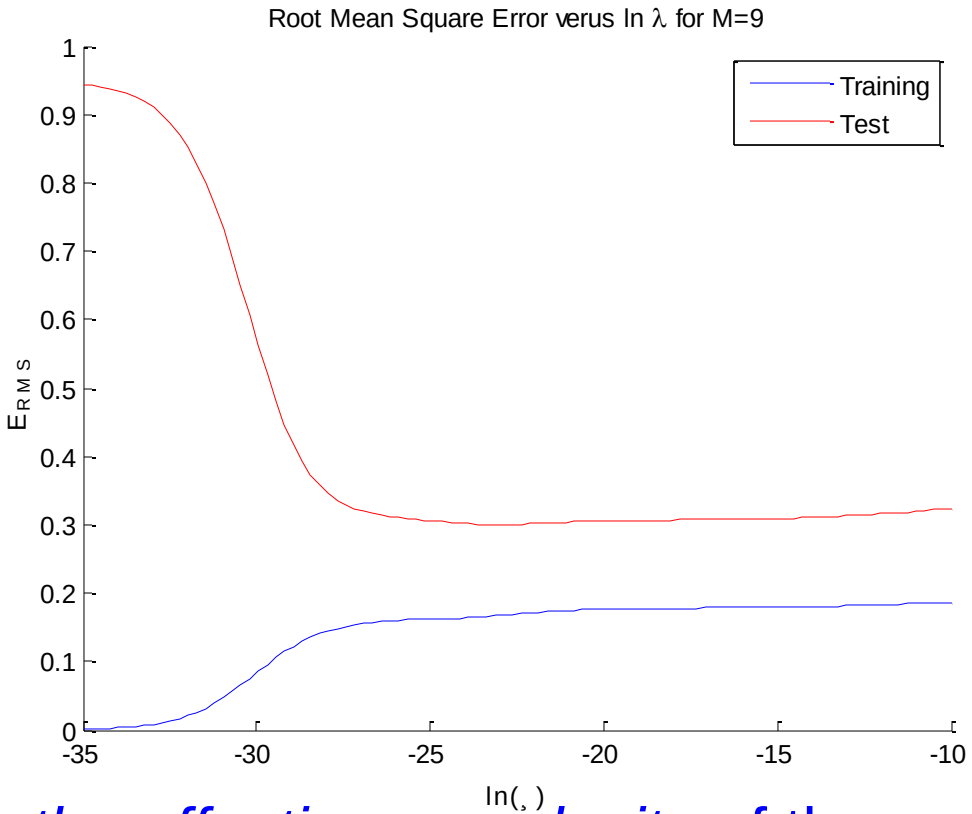
Regularization Controls Model Complexity



MatLab Code

- If, however, we use too large a value for λ then we again obtain a poor fit, as shown for $\ln\lambda = 0$.

Regularization Controls Model Complexity



MatLab Code

- *λ controls the effective complexity* of the model and hence determines the degree of over-fitting.

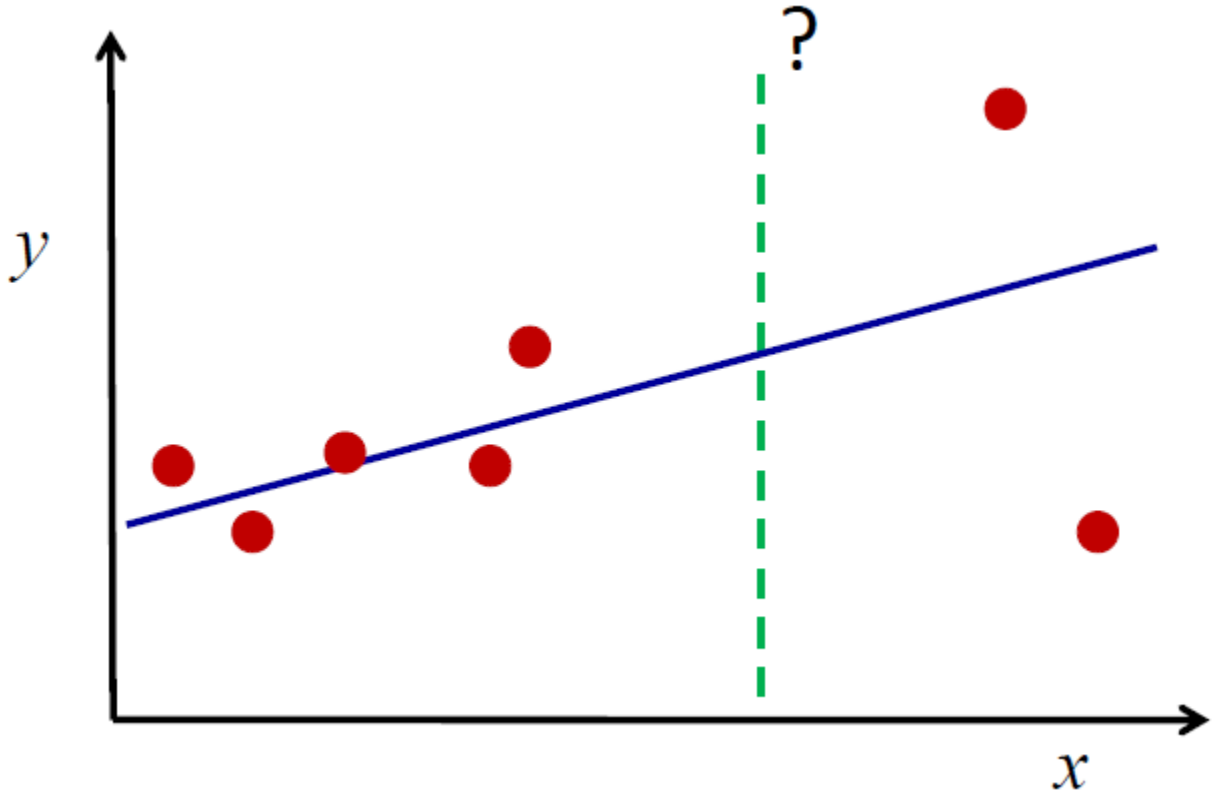
- We will soon re-examine this problem with a Bayesian approach that avoids the over-fitting problem.

MLE, Regularization and Model Complexity

- Effective model complexity in MLE is governed by the number of basis functions and is controlled by the size of the data set.
- With regularization, the effective model complexity is controlled mainly by λ and still by the number and form of the basis functions.
- *Thus the model complexity for a particular problem cannot be decided simply by maximizing the likelihood function as this leads to excessively complex models and over-fitting.*
- Independent hold-out data can be used to determine model complexity but this wastes data and it is computationally expensive.

Prior Knowledge is Essential

- We cannot do everything simply based on data – prior knowledge is essential to inference and prediction.



Frequentist Versus Bayesian Paradigms

- The likelihood $p(\mathcal{D}|\boldsymbol{w})$ is essential in both Bayesian and frequentist approaches but it is used in different roles.
- In a frequentist approach
  - $\boldsymbol{w}$ is a fixed parameter computed by an estimator (e.g. maximum likelihood estimator).
  - Error bars on this point estimate are computed by considering the distribution of all possible data sets $\mathcal{D}$ (e.g. variability of predictions between different bootstrap data sets)
- In Bayesian approach
  - there is only one set of data $\mathcal{D}$ and
  - the uncertainty in $\boldsymbol{w}$ is introduced with appropriate prior and computing posterior probabilities over $\boldsymbol{w}$.

Bayesian Probabilities

- For example in the regression problem with the observed data $\mathcal{D} = \{t_1, \ldots, t_N\}$, we can obtain the conditional probability $p(\boldsymbol{w}|\mathcal{D})$ by Bayes' theorem

$$p(\boldsymbol{w} \mid \boldsymbol{\mathcal{D}}) = \frac{p(\boldsymbol{\mathcal{D}} \mid \boldsymbol{w})\, p(\boldsymbol{w})}{p(\boldsymbol{\mathcal{D}})}, \; p(\boldsymbol{\mathcal{D}}) = \int p(\boldsymbol{\mathcal{D}} \mid \boldsymbol{w})\, p(\boldsymbol{w}) d\boldsymbol{w}$$

- The quantity $p(\mathcal{D}|\boldsymbol{w})$ on the right-hand side of Bayes' theorem is evaluated for the observed data set $\mathcal{D}$ and can be viewed as a function of the parameter vector $\boldsymbol{w}$ (*likelihood function*).

- Given this definition of likelihood, we can state Bayes' theorem in words

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

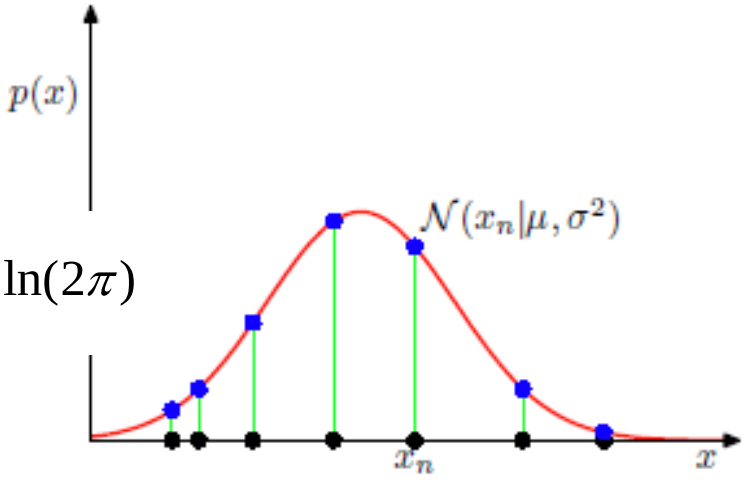
Gaussian Distribution

- Consider the Gaussian distribution

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\left(2\pi\sigma^2\right)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

- The likelihood function for the Gaussian distribution is

$$p(\mathcal{D} \mid \mu, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n \mid \mu, \sigma^2)$$



- The log likelihood takes the form*

$$\ln p(\mathcal{D} \mid \mu, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi)$$

- Maximum likelihood solution

$$\mu_{ML} = \frac{\sum_{n=1}^{N} x_n}{N}, \quad \sigma^2_{ML} = \frac{\sum_{n=1}^{N}(x_n - \mu_{ML})^2}{N}$$

* We work often with log-likelihood to avoid underflow (taking products of small probabilities) and for simplifying the algebra.

MLE for a Gaussian Distribution

$$\mu_{ML} = \underbrace{\frac{1}{N}\sum_{i=1}^{N} x_i}_{\text{Sample mean}}, \quad \sigma^2_{ML} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{ML})^2}_{\text{Sample variance wrt ML mean (not the exact mean)}}$$

- The MLE approach underestimates the variance (bias) – this is in the root of the over-fitting problem e.g. in polynomial curve fitting.
- The maximum likelihood solutions μ_{ML}, σ^2_{ML} are functions of the data set values $x_1, \ldots, x_N$. Consider the expectations of these quantities with respect to the data set values, which come from a Gaussian.
- Using the point estimates above you can show that :

$$\mathbb{E}[\mu_{ML}] = \mu, \quad \mathbb{E}[\sigma^2_{ML}] = \frac{N-1}{N}\sigma^2$$

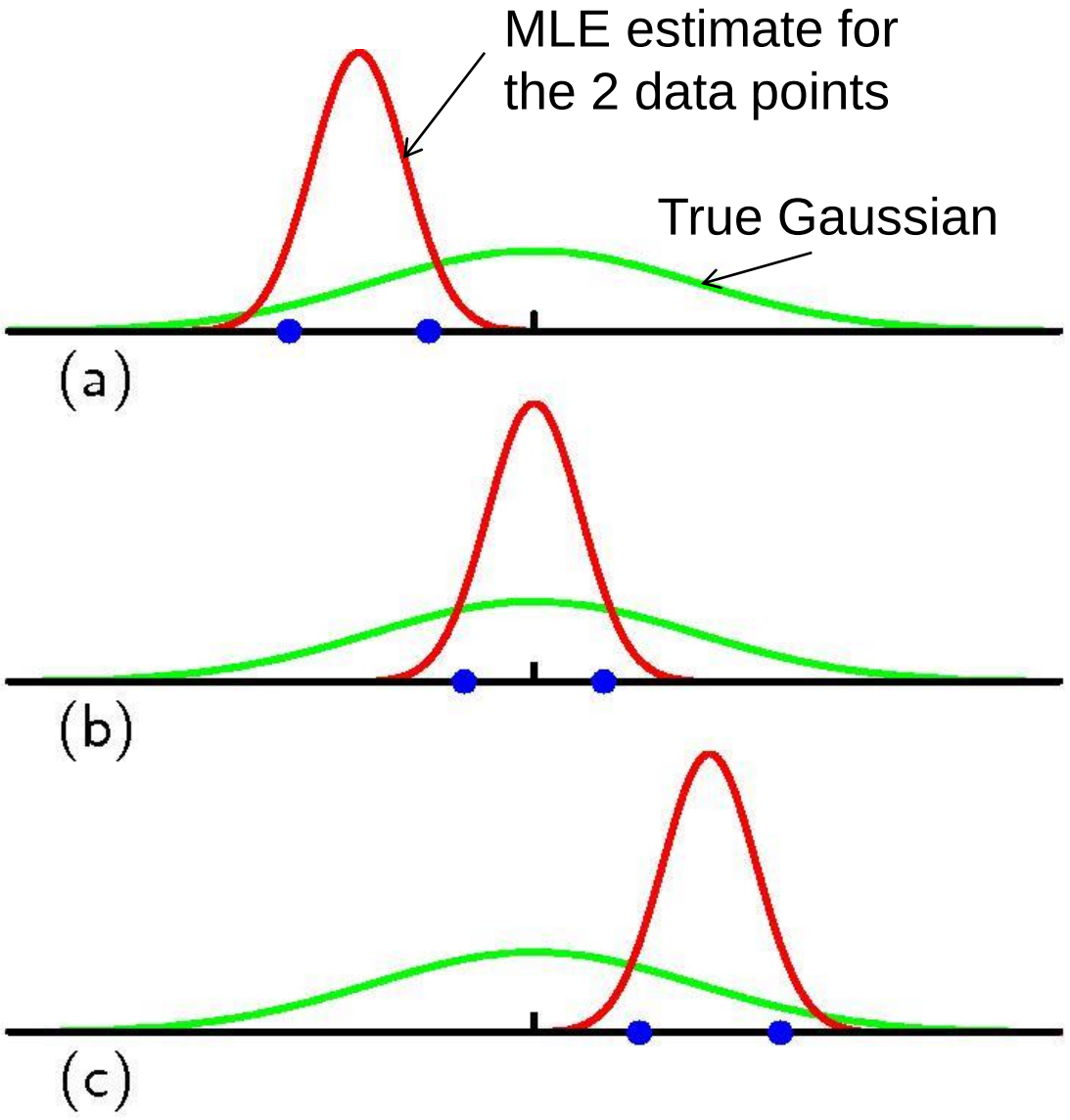
In this derivation you need to use :

$$\mathbb{E}[x_i x_j] = \mu^2 \quad \text{for } i \neq j$$

$$\mathbb{E}[x_i^2] = \sigma^2 + \mu^2$$

MLE: Underestimating the Variance

- In the schematic from Bishop's PRML, we consider 3 cases each with 2 data points extracted from the true Gaussian.
- The mean of the 3 distributions predicted via MLE (i.e. averaged over the data) is correct.
- However, ***the variance is underestimated*** since it is a variance with respect to the sample mean and NOT the true mean.



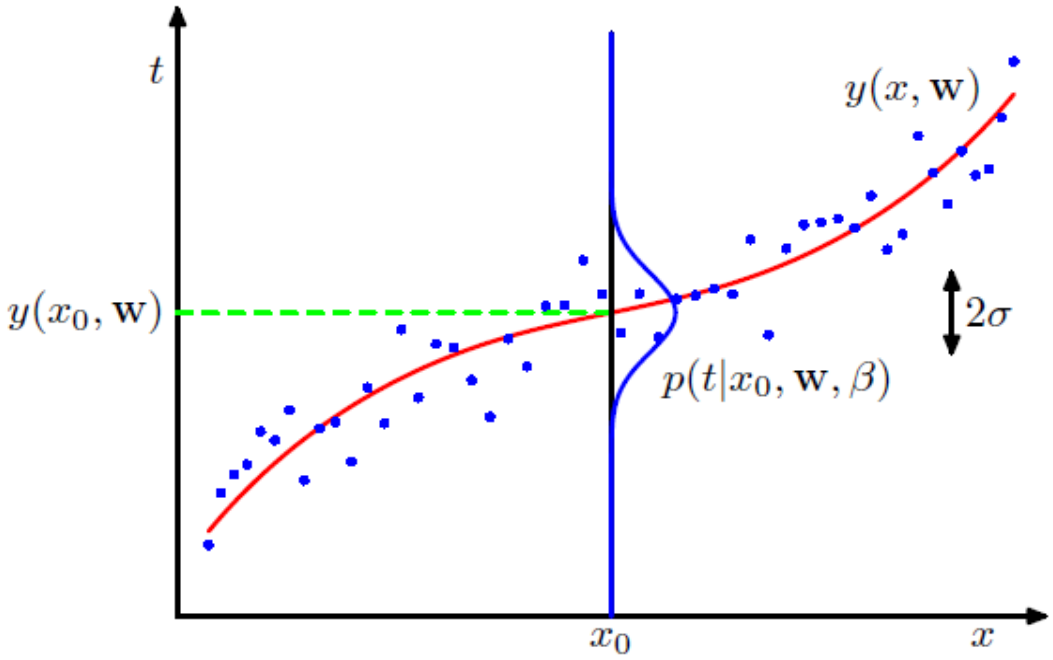
Linear Basis Functions Models

- We are now interested in a linear combination – regression – of a ``fixed set'' of nonlinear basis functions.
- Supervised learning: N observations $\{\boldsymbol{x}_n\}$ with corresponding target values $\{t_n\}$ are provided.
- The goal is to predict t for a new value $\boldsymbol{x}$.
- We construct a function such that $y(\boldsymbol{x})$ is a prediction of t.
- We follow a Bayesian perspective and model the predictive distribution $p(t|\boldsymbol{x})$.

Linear Regression

- From the conditional distribution $p(t|\boldsymbol{x})$, we can make point estimates of t for a given $\boldsymbol{x}$ by minimizing a `loss function'.

- For a quadratic loss function, the point estimate is the conditional mean $y(\boldsymbol{x}, \boldsymbol{w}) = \mathbb{E}[t|\boldsymbol{x}]$.



Linear Regression

- The simplest linear model for regression is one that involves a linear combination of the input variables

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

where $\boldsymbol{x} = (x_1 \quad x_2 \quad . \quad . \quad x_D)^T$.

- This is often simply known as linear regression.
- D is the input dimensionality.
- This model is a linear function of the parameters.

Linear Basis Functions Models

- More generally regression function is:

$$y(\boldsymbol{x},\boldsymbol{w}) = w_0 + w_1\phi_1(\boldsymbol{x}) + \ldots + w_{M-1}\phi_{M-1}(\boldsymbol{x}) = \sum_{i=0}^{M-1} w_i\phi_i(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})$$

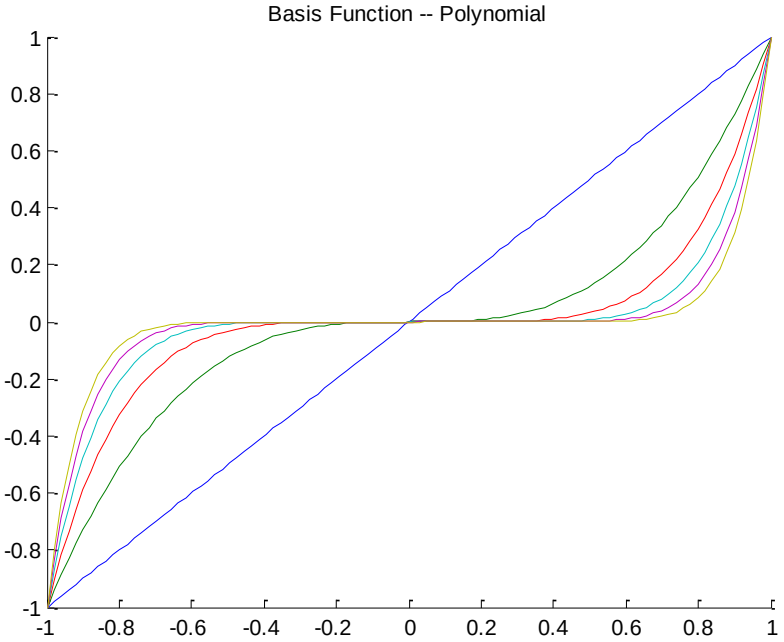
where $\phi_i(\boldsymbol{x})$ are known as basis functions and

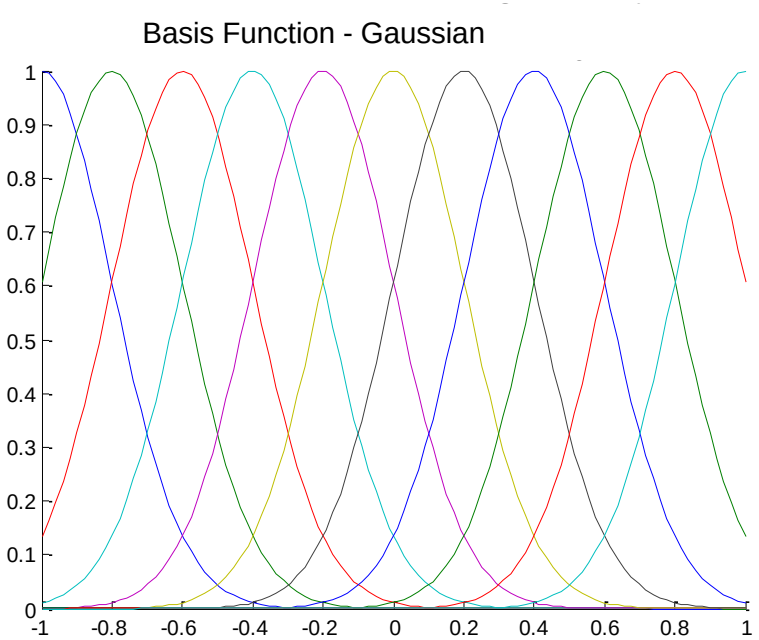
$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(\phi_0(\boldsymbol{x}) \quad \phi_1(\boldsymbol{x}) \quad \phi_2(\boldsymbol{x}) \quad . \quad . \quad \phi_{M-1}(\boldsymbol{x})\right)^T, \phi_0(\boldsymbol{x}) = 1, w_0 = bias$$

- Often $\{\phi_i(\boldsymbol{x})\}$ represent features extracted from the data .

$\boldsymbol{x}$

Polynomial and Gaussian Basis Functions





MatLab code

$$\mu = -1:0.2:1$$

$$s = 0.2$$

Polynomial basis functions (scalar input, global support):

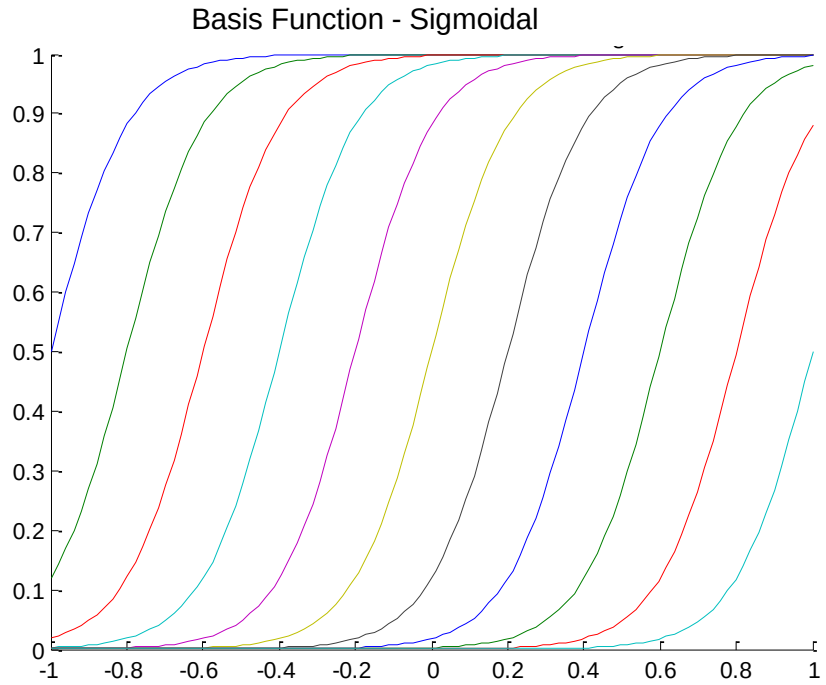
$$\phi_j(x) = x^j$$

Gaussian basis functions:

$$\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right)$$

Logistic Sigmoidal Basis Functions

$$\mu = -1 : 0.2 : 1$$
$$s = 0.1$$



MatLab code

Sigmoidal basis functions:

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right), \quad \sigma(a) = \frac{1}{1+e^{-a}}$$

$$\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}} = 2\sigma(2a) - 1$$

$\sigma(a)$: *logistic sigmoid function*

Sigmoidal and Tanh Basis Functions

- The sigmoidal and tanh basis functions are related:

$$\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} = 2\sigma(2a) - 1 \quad \text{where} \quad \sigma(a) = \frac{1}{1 + e^{-a}}$$

- *A general linear combination of logistic sigmoidal functions is equivalent to a linear combination of* tanh *functions*:

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \sigma\left(\frac{x - \mu_j}{s}\right) = w_0 + \sum_{j=1}^{M-1} w_j \sigma\left(2\frac{x - \mu_j}{2s}\right)$$

$$= w_0 + \sum_{j=1}^{M-1} w_j \frac{1 + \tanh\left(\frac{x - \mu_j}{2s}\right)}{2} = u_0 + \sum_{j=1}^{M-1} u_j \tanh\left(\frac{x - \mu_j}{2s}\right)$$

$$where: \; u_0 = w_0 + \frac{1}{2}\sum_{j=1}^{M-1} w_j, \; u_j = \frac{w_j}{2}$$

Choice of Basis Functions

- We are interested in functions of ***local support*** to explore adaptivity.
- Local support functions comprise a spectrum of different spatial frequencies.
- An example is wavelets that are local both spatially and in frequency.
  - They are however useful only when the input is defined on a lattice.

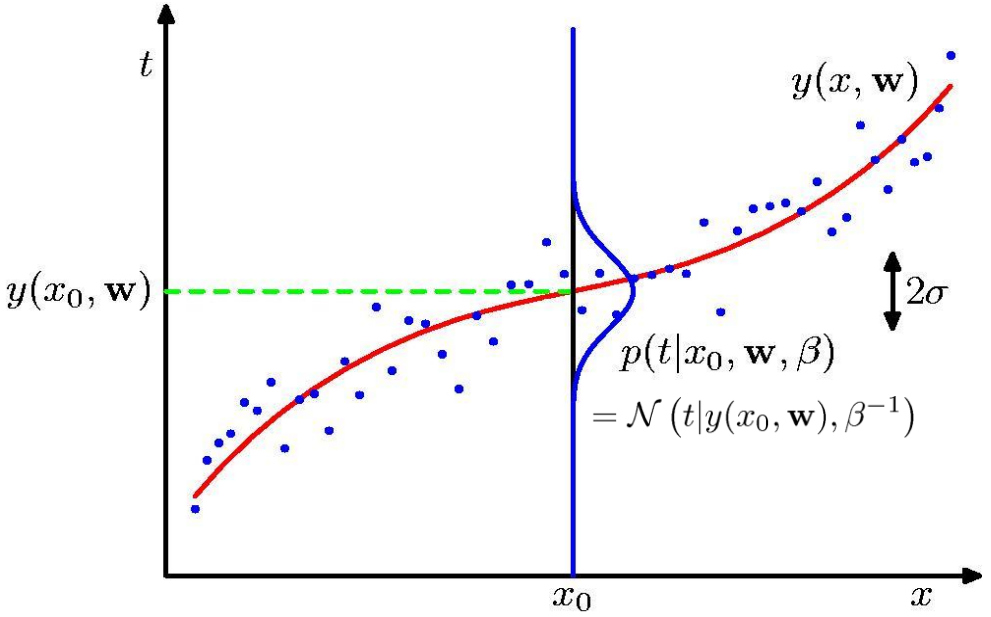
MLE and Least Squares

- Consider a set of training data comprising N input $\boldsymbol{x} = (x_1, \ldots, x_N)^T$ & the corresponding target values $\boldsymbol{t} = (t_1, \ldots, t_N)^T$

- We assume that, given the value of x, the corresponding value of t has a Gaussian distribution with a mean equal to the value $y(x, \boldsymbol{w})$ and precision (inverse of the variance) β

$$p(t \mid x, \boldsymbol{w}, \beta) = \mathcal{N}(t \mid y(x, \boldsymbol{w}), \beta^{-1})$$

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{w}$$
$$\mathrm{t} = y(\boldsymbol{x}, \boldsymbol{w}) + \varepsilon$$
$$\varepsilon \sim \mathcal{N}(0, \beta^{-1})$$



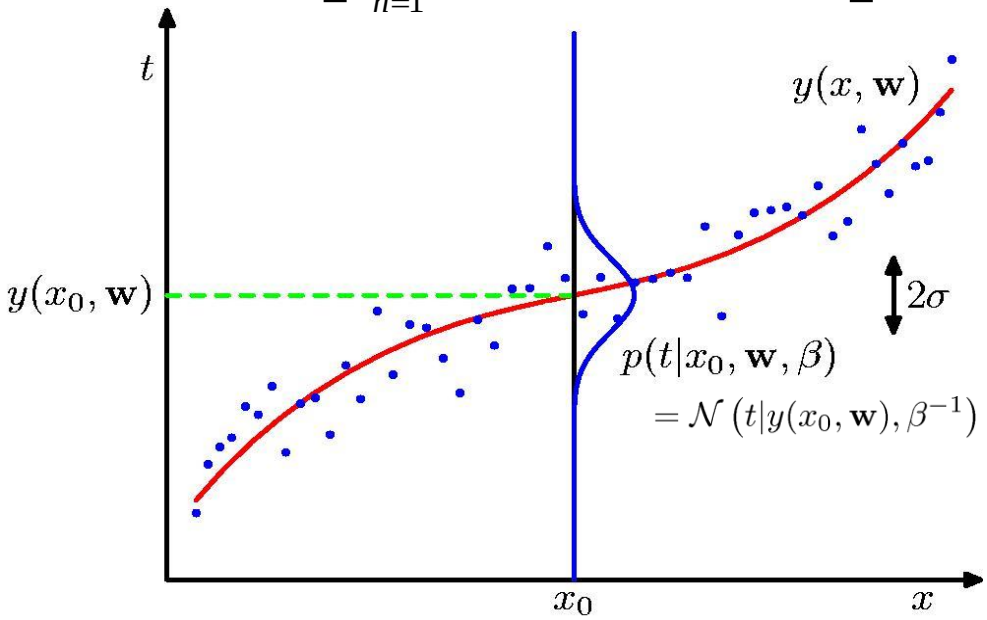
MLE and Least Squares

- The likelihood function is

$$p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n \mid y(x_n, \boldsymbol{w}), \beta^{-1})$$

- From this, the log-likelihood takes the form:

$$\ln p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{w}, \beta) = -\frac{\beta}{2} \sum_{n=1}^{N} \{y(x_n, \boldsymbol{w}) - t_n\}^2 + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi)$$



MLE and Least Squares

- Consider first the MLE estimate for $\boldsymbol{w}$. Note that maximizing the log-likelihood to obtain $\boldsymbol{w}_{ML}$ is the same as minimizing the sum of squares error function (residual sum of squares, $RSS(\boldsymbol{w})$):

$$\max_{\boldsymbol{w}} \ln p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{w}, \beta) = -\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \boldsymbol{w}) - t_n\}^2 + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) \Leftrightarrow$$

$$\boldsymbol{w}_{ML} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \boldsymbol{w}) - t_n\}^2$$

- We can also determine the MLE estimate of β:

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \boldsymbol{w}_{ML}) - t_n\}^2$$

- We can now make (a frequentist, plug-in approximation) prediction as follows: $p(t \mid x, \boldsymbol{w}_{ML}, \beta_{ML}) = \mathcal{N}(t \mid y(x, \boldsymbol{w}_{ML}), \beta_{ML}^{-1})$

Maximum Likelihood and Least Squares

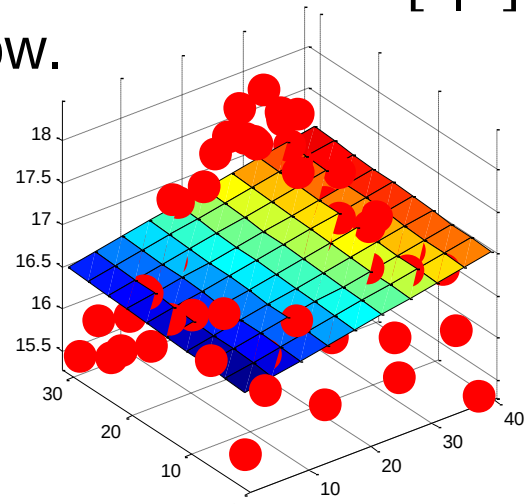
- Assume observations from a deterministic function with added Gaussian noise:

$$t = y(\boldsymbol{x}, \boldsymbol{w}) + \varepsilon \qquad \text{where} \qquad p(\varepsilon \mid \beta) = \mathcal{N}(\varepsilon \mid 0, \beta^{-1})$$

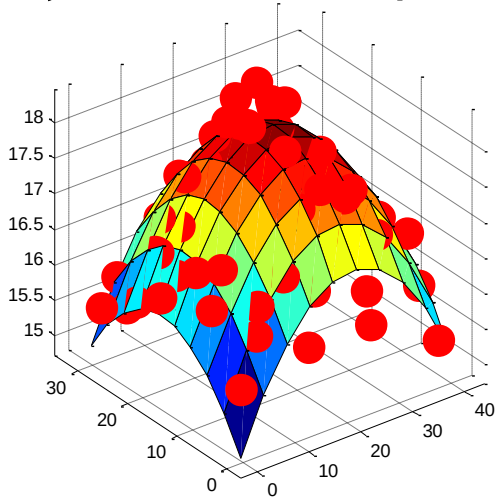
which is the same as saying,

$$p(t \mid \boldsymbol{x}, \boldsymbol{w}, \beta) = \mathcal{N}(t \mid y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1})$$

- Here β is the precision. This is based on a squared loss function for which $\mathbb{E}[t|\boldsymbol{x}] = y(\boldsymbol{x}, \boldsymbol{w})$. A 2D example is shown below.



$$\mathbb{E}[t \mid \boldsymbol{x}] = w_0 + w_1 x_1 + w_2 x_2$$



$$\mathbb{E}[t \mid \boldsymbol{x}] = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2$$

Run surfaceFitDemo from PMTK3

Maximum Likelihood and Least Squares

- Given observed inputs, $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, and targets $\boldsymbol{t} = \{t_1, ..., t_N\}^T$, we obtain the likelihood function

$$p(\boldsymbol{t} \mid \boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}\left(t_n \mid \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\right)$$

- We often use the *log-likelihood*:

$$\ell(\boldsymbol{w}, \beta) = \log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{n=1}^{N} \log \mathcal{N}\left(t_n \mid \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\right), \boldsymbol{\theta} = (\boldsymbol{w}, \beta)$$

- Instead of maximizing the log-likelihood, one equivalently can minimize *the negative log-likelihood (NLL)*

$$NLL(\boldsymbol{w}, \beta) = -\sum_{n=1}^{N} \log \mathcal{N}\left(t_n \mid \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\right)$$

Maximum Likelihood and Least Squares

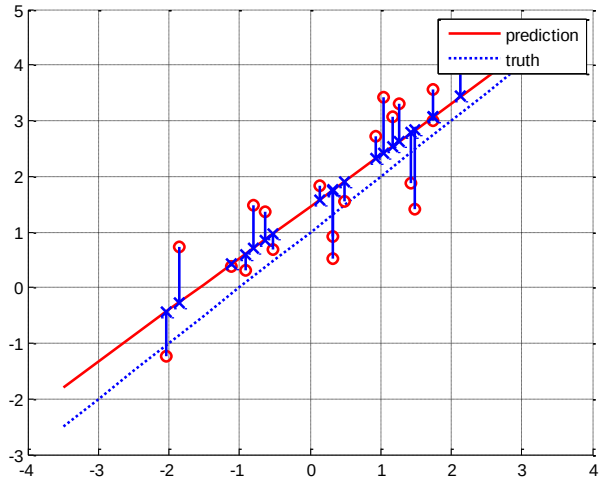
- Taking the log of the likelihood, we obtain:

$$\ln p(\boldsymbol{t} \mid \boldsymbol{w}, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n \mid \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\right) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\boldsymbol{w})$$

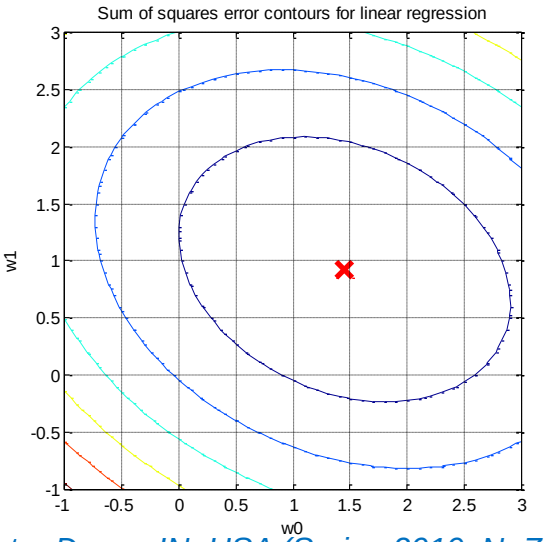
where with $E_D(\boldsymbol{w})$ we have denoted:

$$E_D(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2, \; RSS = \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2, \; MSE = RSS / N$$

- RSS is often known as the *residual sum of squares* or *sum of squared errors (SSE)* and MSE *is the mean squared error*.
- Computing $\boldsymbol{w}$ via *MLE* is the same as *Least Squares*.



Run residualsDemo from PMTK3



The NLL is a quadratic bowl with a unique minimum (the MLE estimate)

Run contoursSSEdemo, from PMTK3

Maximum Likelihood and Least Squares

$$\ln p(\boldsymbol{t} \mid \boldsymbol{w}, \beta) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln (2\pi) - \beta E_D(\boldsymbol{w}), \; E_D(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) \right)^2$$

- Setting the gradient of the log-likelihood (written here as a row vector) wrt $\boldsymbol{w}$ equal to zero:

$$\nabla_{\boldsymbol{w}} \ln p(\boldsymbol{t} \mid \boldsymbol{w}, \beta) = \beta \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) \right) \boldsymbol{\phi}(\boldsymbol{x}_n)^T = \beta \left\{ \sum_{n=1}^{N} t_n \boldsymbol{\phi}(\boldsymbol{x}_n)^T - \boldsymbol{w}^T \sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_n) \boldsymbol{\phi}(\boldsymbol{x}_n)^T \right\} = 0$$

- This equation can be solved for $\boldsymbol{w}$.

Maximum Likelihood and Least Squares

$$\boldsymbol{w}^T \sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T = \sum_{n=1}^{N} t_n \boldsymbol{\phi}(\boldsymbol{x}_n)^T \Rightarrow \boldsymbol{w}^T \boldsymbol{\Phi}^T \boldsymbol{\Phi} = \left(\boldsymbol{\Phi}^T \boldsymbol{t}\right)^T \Rightarrow \boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{w} = \boldsymbol{\Phi}^T \boldsymbol{t}$$

- We obtain (normal equation; ordinary least squares solution):

$$\boldsymbol{w}_{ML} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \equiv \boldsymbol{\Phi}^{\dagger} \boldsymbol{t}, \quad \boldsymbol{\Phi}^{\dagger} : \text{Moore-Penrose pseudo-inverse}$$

where we have defined:

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}^T(\boldsymbol{x}_1) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{\phi}^T(\boldsymbol{x}_N) \end{pmatrix} = \begin{pmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & .. & \phi_{M-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & .. & \phi_{M-1}(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & .. & \phi_{M-1}(\boldsymbol{x}_N) \end{pmatrix},$$

$$\boldsymbol{\Phi}^T = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_2) & .. & \boldsymbol{\phi}(\boldsymbol{x}_N) \end{pmatrix}$$

$$\boldsymbol{\phi}(\boldsymbol{x}_i) \equiv \begin{pmatrix} \phi_0(\boldsymbol{x}_i) & \phi_1(\boldsymbol{x}_i) & . & . & \phi_{M-1}(\boldsymbol{x}_i) \end{pmatrix}^T$$

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\varphi}_0 & \boldsymbol{\varphi}_1 & .. & \boldsymbol{\varphi}_{M-1} \end{pmatrix}$$

$$\boldsymbol{\varphi}_i \equiv \begin{pmatrix} \phi_i(\boldsymbol{x}_1) & \phi_i(\boldsymbol{x}_2) & . & . & \phi_i(\boldsymbol{x}_N) \end{pmatrix}^T$$

- Note that indeed:

$$\boldsymbol{\Phi}^T \boldsymbol{\Phi} = \sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T, \quad \boldsymbol{\Phi}^T \boldsymbol{t} = \sum_{n=1}^{N} t_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

Maximum Likelihood and Least Squares

$$\ln p\left(\boldsymbol{t} \mid \boldsymbol{w}, \beta\right) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n \mid \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\right) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln\left(2\pi\right) - \beta E_D(\boldsymbol{w})$$

- Taking now the log of the likelihood with respect to β gives:

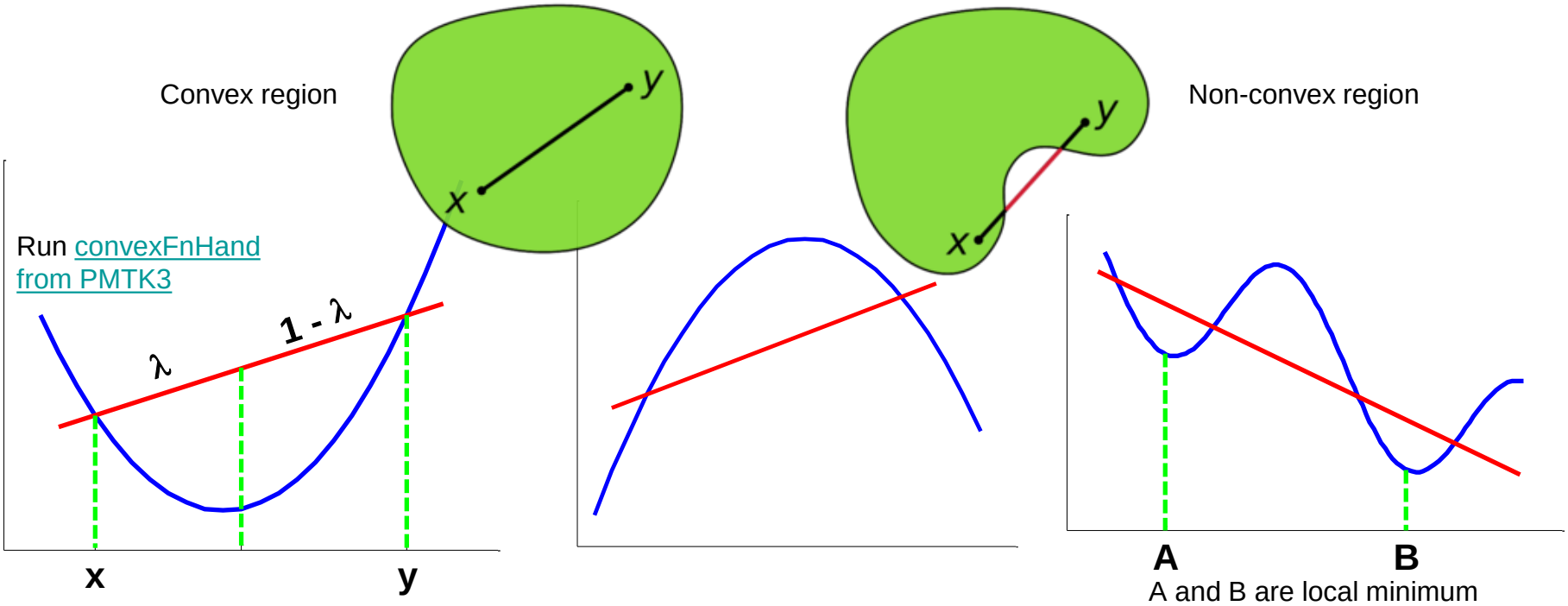
$$\frac{1}{\beta_{ML}} = \frac{2}{N} E_D(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2$$

- So the MLE variance β_{ML} is equal to the residual variance of the target values around the regression function.

Convexity of the NLL

- *Convexity of the NLL (positive definite Hessian)* leads to a unique globally optimal MLE.

- Some models of interest don't have concave likelihoods and locally optimal MLE estimates are needed.



Sequential Learning: LMS Algorithm

- If the data set is large, we use sequential (on-line) algorithms
- We apply the technique of ***stochastic (sequential) gradient descent***.
- If the error function comprises a sum over data points $E = \sum_n E_n$

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2$$

then after presentation of pattern n, the stochastic gradient descent algorithm updates the parameter vector $\boldsymbol{w}$ using

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E_n = \boldsymbol{w}^{(\tau)} + \eta\underbrace{\left(t_n - \boldsymbol{w}^{(\tau)^T}\boldsymbol{\phi}(\boldsymbol{x}_n)\right)}_{-\nabla E_n}\boldsymbol{\phi}(\boldsymbol{x}_n)$$

τ is the iteration number & η the learning rate parameter.

- This is known as ***least-mean-squares*** *or the* ***LMS algorithm***.

Robust Linear Regression

- Using a Gaussian distribution for the noise,

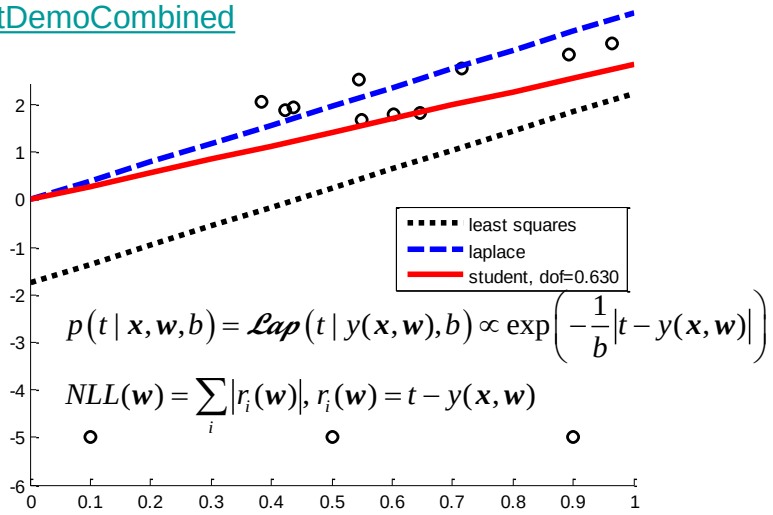
$$t = y(\boldsymbol{x}, \boldsymbol{w}) + \varepsilon,\ \varepsilon \sim \mathcal{N}\left(\varepsilon \mid 0, \beta^{-1}\right)$$

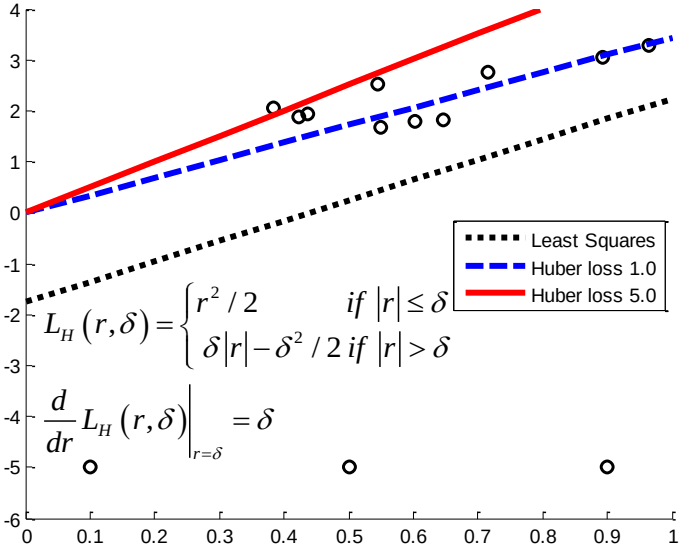
  can result in poor fit especially if we have outliers in the data.

- Squared error penalizes deviations quadratically, so points far from the line have more affect on the fit than points near the line.

- *To achieve robustness to outliers one can replace the Gaussian with a distribution that has heavy tails (e.g. the Laplace distribution).* Such a distribution assigns higher likelihood to outliers, without having to perturb the regression line to "explain" them.

Run linregRobustDemoCombined from PMTK3





Robust Linear Regression

- Using the Laplace distribution leads to L_1 error norm (non-linear objective function) that is difficult to optimize.

- A solution is to transform the problem (by increasing its dimension to $2N + M$) to a *linear programming problem*.

$$r_i \triangleq r_i^+ - r_i^-$$

$$\min_{\boldsymbol{w}, r_i^+, r_i^-} \sum_i \left(r_i^+ + r_i^- \right) \quad s.t. \;\; r_i^+ \geq 0, r_i^- \geq 0, \boldsymbol{w}^T \boldsymbol{x}_i + r_i^+ - r_i^- = t_i$$

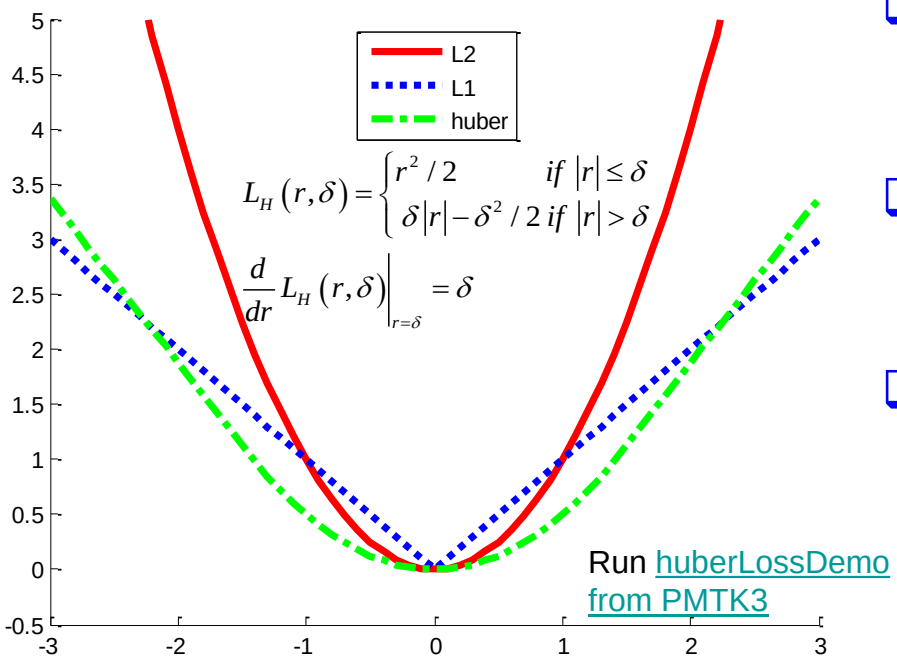
- Note that with our definition above:

$$r_i^+ = \frac{1}{2}(r_i + |r_i|) = \begin{cases} r_i \; if \;\; r_i \geq 0 \\ 0 \; if \;\; r_i < 0 \end{cases}, \quad r_i^- = \frac{1}{2}(|r_i| - r_i) = \begin{cases} 0 \quad if \;\; r_i > 0 \\ -r_i \; if \;\; r_i \leq 0 \end{cases}$$

$$|r_i| = r_i^+ + r_i^-$$

- Boyd, S. and L. Vandenberghe (2004). *Convex optimization*. Cambridge

Huber Loss Function



Run huberLossDemo from PMTK3

- *This is equivalent to L_2 for errors that are smaller than δ, and is equivalent to L_1 for larger errors.*
- This loss function is everywhere differentiable, using the fact that $d/dr\,|r| = sign(r)\ if\ r \neq 0$.
- The function is also C_1 continuous, since the gradients of the two parts of the function match at $r = \pm\delta$.

- Optimizing the Huber loss is much faster than using the Laplace likelihood, since we can use standard optimization methods (quasi-Newton) rather than linear programming.
- The Huber method also has a probabilistic interpretation, although it is rather unnatural (Pontil et al. 1998).

- Pontil, M., S. Mukherjee, and F. Girosi (1998). On the Noise Model of Support Vector Machine Regression. Technical report, MIT AI Lab.
- Huber, P. (1964). Robust estimation of a location parameter. *Annals of Statistics 53*, 73-101.

Regularized LS – Ridge Regression

- Consider the error function:

$$E_D(\boldsymbol{w}) + \lambda E_W(\boldsymbol{w})$$

data term + regularization term

- With the sum-of-squares error function and a quadratic regularizer, we get

$$\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2 + \lambda\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$

- Specifically, setting the gradient with respect to $\boldsymbol{w}$ to zero, and solving for $\boldsymbol{w}$ as before, we obtain

$$\boldsymbol{w} = \left(\lambda\boldsymbol{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$$

- This is a trivial extension of the least-squares solution we encountered earlier (*Regularized Least Squares – Ridge Regression*)

Regularized Least Squares

- Regularized solution:

$$\boldsymbol{w} = \left(\lambda \boldsymbol{I} + \boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}$$

- Regularization limits the effective model complexity (the appropriate number of basis functions).

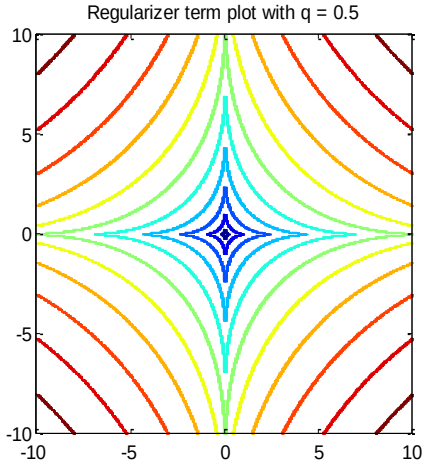
- This is replaced with the problem of finding a suitable value of the regularization coefficient λ.

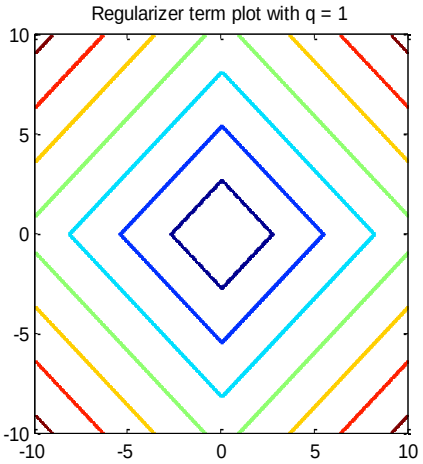
- *λ controls how many non-zero $\boldsymbol{w}$'s (i.e. basis functions) you have.*

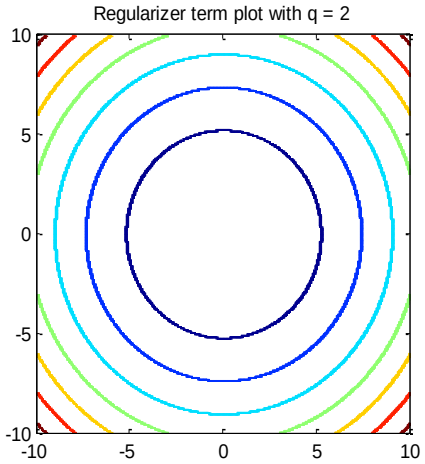
Regularized Least Squares

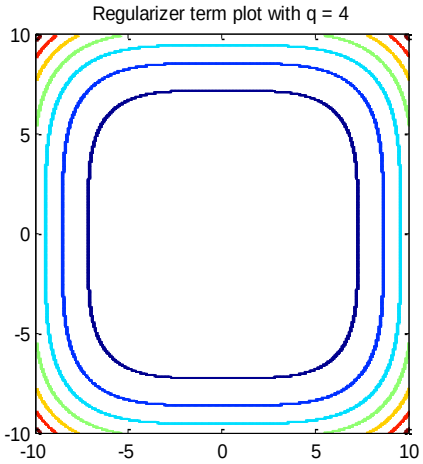
- With a more general regularizer, we have

$$\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2 + \lambda\frac{1}{2}\sum_{j=1}^{M}\left|w_j\right|^q$$









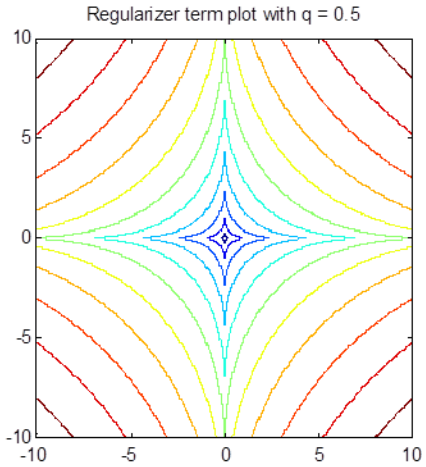
MatLab code

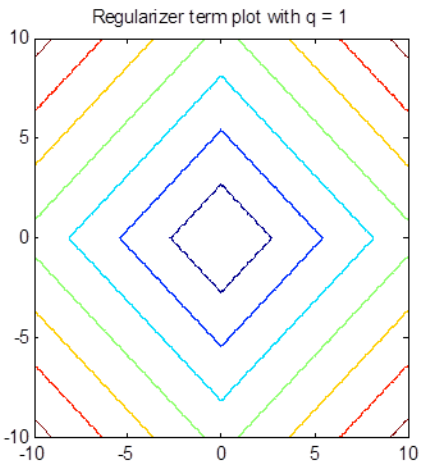
- $q = 1$ is known as the Lasso regularizer. These plots show only the regularizer term with $\lambda = 0.7334$.

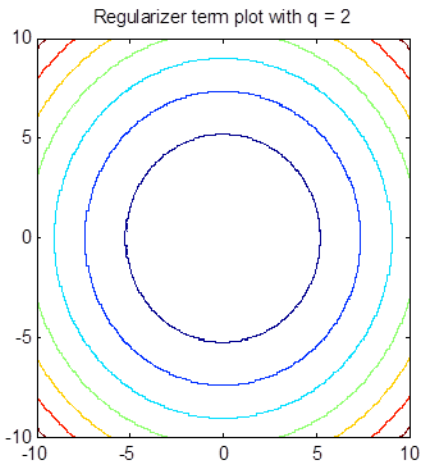
Regularized Least Squares

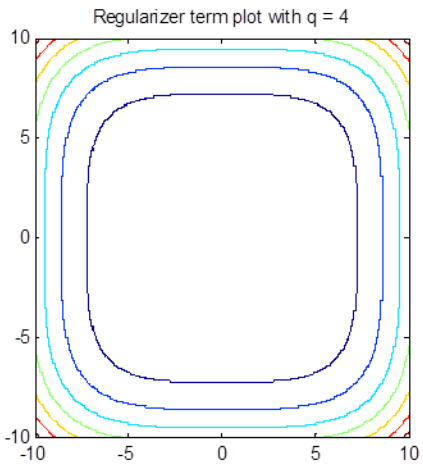
- With a more general regularizer, we have

$$\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2 + \lambda\frac{1}{2}\sum_{j=1}^{M}\left|w_j\right|^q$$







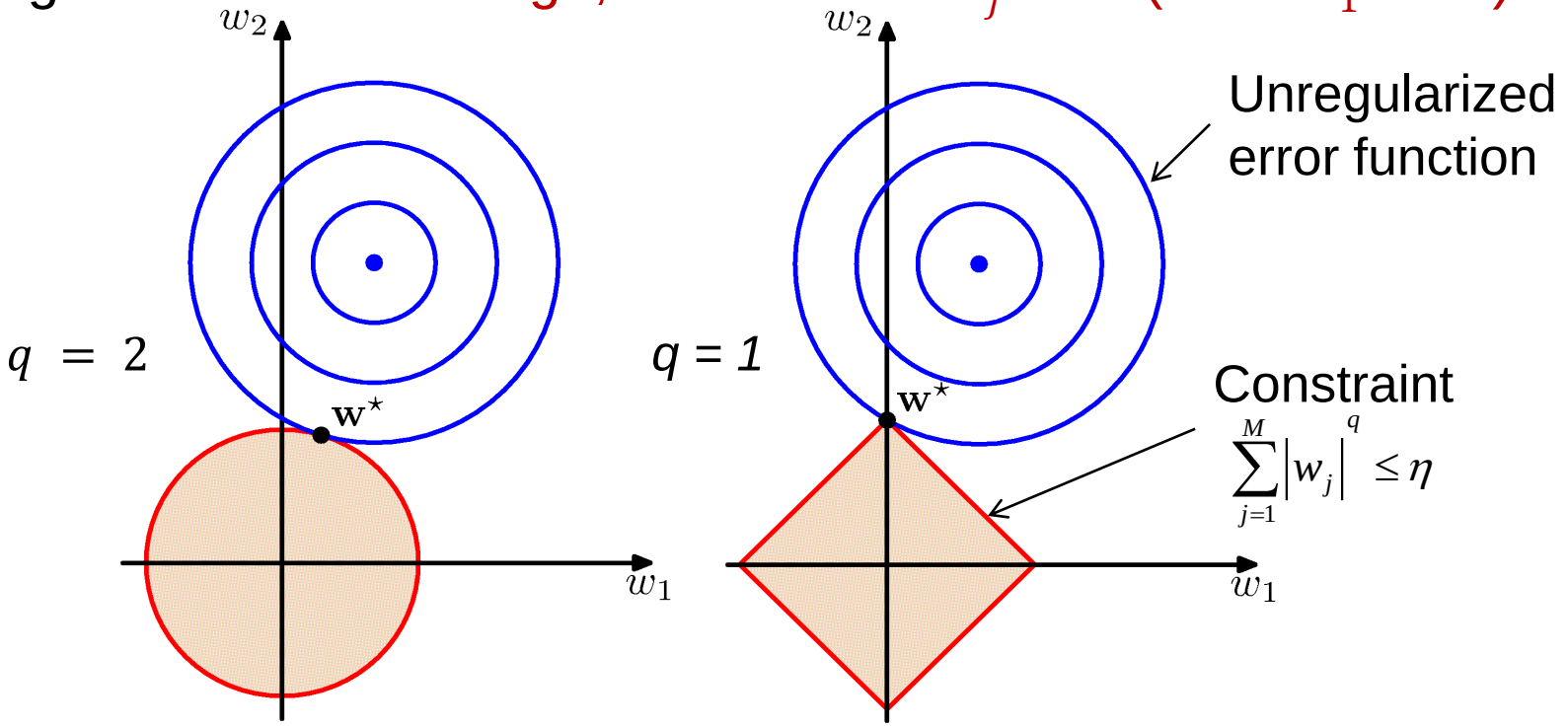


MatLab code

- $q = 2$ corresponds to the quadratic regularizer.

Regularized Least Squares

- Lasso tends to generate sparser solutions than a quadratic regularizer – if λ is large, some of the $w_j \rightarrow 0$ (here $w_1 = 0$).



- Here, we consider that the regularized least squares solution is equivalent to minimizing the unregularized sum of squares with the constraint shown for some η (see proof next).

Regularized Least Squares

- Let us write the constraint in the equivalent form: $\frac{1}{2}\left(\sum_{j=1}^{M}\left|w_j\right|^q - \eta\right) \leq 0$

- This leads to the following Lagrangian function:

$$L(\boldsymbol{w},\lambda) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2 + \frac{\lambda}{2}\left(\sum_{j=1}^{M}\left|w_j\right|^q - \eta\right)$$

- This is identical to our regularized least squares (RLS) in the dependence on $\boldsymbol{w}$.

$$\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2 + \lambda\frac{1}{2}\sum_{j=1}^{M}\left|w_j\right|^q \quad (*)$$

- For a particular $\lambda > 0$, let $\boldsymbol{w}^*(\lambda)$ be the solution of the RLS in (*).

- From the Kuhn-Tucker optimality conditions for $L(\boldsymbol{w},\lambda)$ we then see:

$$\eta = \sum_{j=1}^{M}\left|w_j^*\right|^q$$

Kuhn-Tucker Optimality Conditions

- Consider the following constraint minimization problem:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}), \textit{subject to } g(\boldsymbol{x}) \geq 0$$

- This is equivalent as the minimization with respect to $\boldsymbol{x}$ and λ of the following Lagrangian:

$$\min_{\boldsymbol{x},\lambda} L(\boldsymbol{x},\lambda) = f(\boldsymbol{x}) - \lambda\, \mathrm{g}(\boldsymbol{x})$$

subject to the following (Kuhn-Tucker) conditions:

$$\lambda \geq 0,\ \mathrm{g}(\boldsymbol{x}) \geq 0,\ \lambda\, \mathrm{g}(\boldsymbol{x}) = 0$$

- Note for maximization problems, the Lagrangian should be modified as: $L(\boldsymbol{x},\lambda) = f(\boldsymbol{x}) + \lambda\, \mathrm{g}(\boldsymbol{x})$

Multiple Outputs-Isotropic Covariance

- If we want to predict $K > 1$ target variables, we use the same basis for all components of the target vector):

$$p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{W}, \beta) = \mathcal{N}\left(\boldsymbol{t} \mid \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{W}), \beta^{-1}\boldsymbol{I}\right) = \mathcal{N}\left(\boldsymbol{t} \mid \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}), \beta^{-1}\boldsymbol{I}\right)$$

$\boldsymbol{W}$ is an $M \times K$ matrix and $\boldsymbol{t}$ is K dimensional.

- Given observed inputs, $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, and targets, $\boldsymbol{T} = \{\boldsymbol{t}_1, ..., \boldsymbol{t}_N\}^T$ we obtain the log likelihood function

$$\ln p(\boldsymbol{T} \mid \boldsymbol{X}, \boldsymbol{W}, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n \mid \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\boldsymbol{I}\right) = \frac{NK}{2}\ln\frac{\beta}{2\pi} - \frac{\beta}{2}\sum_{n=1}^{N}\left\|\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right\|^2$$

K-Independent Regression Problems

$$\ln p\left(\boldsymbol{T} \mid \boldsymbol{X}, \boldsymbol{W}, \beta\right) = \frac{NK}{2}\ln\frac{\beta}{2\pi} - \frac{\beta}{2}\sum_{n=1}^{N}\left\|\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right\|^2$$

- As before, we can maximize this function with respect to $\boldsymbol{W}$, giving

$$\underset{M\times K}{\boldsymbol{W}_{ML}} = \underbrace{\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}}_{M\times M}\underset{M\times N}{\boldsymbol{\Phi}^T}\ \underset{N\times K}{\boldsymbol{T}}$$

- If we examine this result for each target variable t_k, we have (take the k^{th} column of $\boldsymbol{W}$ and $\boldsymbol{T}$):

$$\boldsymbol{w}_{k_{ML}} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}_k = \boldsymbol{\Phi}^{\dagger}\boldsymbol{t}_k$$

  which is identical with the single output case (so there is decoupling between the target variables)

- As expected, we obtain K –independent regression problems.

Multiple Outputs – Full Covariance

- Let us repeat the earlier formulation but with covariance matrix $\boldsymbol{\Sigma}$. If we want to predict $K > 1$ target variables, we use the same basis for all components of the target vector):

$$p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{W}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{t} \mid \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{W}), \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{t} \mid \boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\Sigma})$$

where $\boldsymbol{W}$ is an $M \times K$ matrix of parameters

- Given observed inputs, $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, and targets, $\boldsymbol{T} = \{\boldsymbol{t}_1, ..., \boldsymbol{t}_N\}^T$ we obtain the log likelihood function $\ln p(\boldsymbol{T} \mid \boldsymbol{X}, \boldsymbol{W}, \boldsymbol{\Sigma})$:

$$\sum_{n=1}^{N} \ln \mathcal{N}(t_n \mid \boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \boldsymbol{\Sigma}) = -\frac{N}{2} \ln |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{n=1}^{N} (\boldsymbol{t}_n - \boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x}_n))^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{t}_n - \boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x}_n))$$

Multiple Outputs – Full Covariance

$$\ln p(\boldsymbol{T} \mid \boldsymbol{X}, \boldsymbol{W}, \boldsymbol{\Sigma}) = -\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^T\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)$$

- As before, we maximize this function with respect to $\boldsymbol{W}$,

$$0 = -\sum_{n=1}^{N}\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)\boldsymbol{\phi}(\boldsymbol{x}_n)^T \Rightarrow \underset{M\times K}{\boldsymbol{W}_{ML}} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{T}$$

- For the ML estimate for $\boldsymbol{\Sigma}$, use the result for the MLE of the covariance of a multivariate Gaussian:

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n - \boldsymbol{W}_{ML}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)\left(\boldsymbol{t}_n - \boldsymbol{W}_{ML}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^T$$

- Note that each column of $\boldsymbol{W}_{ML}$ is of the form

$$\boldsymbol{w}_{ML} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$$

seen for isotropic noise distribution and is independent of $\boldsymbol{\Sigma}$!

The Bias-Variance Decomposition

- MLE (i.e. least squares) leads to severe over-fitting if complex models are trained using data sets of limited size.
- Over-fitting occurs whenever the number of basis functions is large (i.e. for complex models) and the training data set is of limited size.
- Limiting the number of basis functions limits the flexibility of the model.
- Regularization controls over-fitting but one needs to determine λ.
- Over-fitting is property of MLE and does not arise when we marginalize over parameters in a Bayesian setting.
- Before returning to a Bayesian setting, we will discuss the bias-variance tradeoff (a frequentist viewpoint of model complexity).

Loss Function and the Regression Function

- Recall the regression loss-function

$$L\big(t, y(\boldsymbol{x})\big) = \big(y(\boldsymbol{x}) - t\big)^2$$

- The decision problem is to minimize the expected loss:

$$\mathbb{E}[L] = \int\int (y(\boldsymbol{x}) - t)^2 p(\boldsymbol{x}, t) d\boldsymbol{x} dt$$

- The solution of this minimization problem known as the regression function is:

$$y(\boldsymbol{x}) = \int t p(t|\boldsymbol{x}) dt = \mathbb{E}_t[t|\boldsymbol{x}]$$

i.e. the average of t conditioned on $\boldsymbol{x}$.

- A useful expression was derived in an earlier lecture:

$$\mathbb{E}[L] = \int (y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2 p(\boldsymbol{x}) dx + \int\int (\mathbb{E}[t|\boldsymbol{x}] - t)^2 p(\boldsymbol{x}, t) d\boldsymbol{x}\, dt$$

The Bias-Variance Decomposition

- We can write the expected squared loss as:

$$\mathbb{E}[L] = \int \big(y(\boldsymbol{x}) - h(\boldsymbol{x})\big)^2 p(\boldsymbol{x}) d\boldsymbol{x} + \iint (h(\boldsymbol{x}) - t)^2 p(\mathbf{x}, t) d\boldsymbol{x} dt$$

where $h(\boldsymbol{x})$ is the conditional expectation which is given by

$$h(\boldsymbol{x}) = \mathbb{E}[t|\boldsymbol{x}] = \int t p(t|\boldsymbol{x}) dt$$

- Recall that the second term, which is independent of $y(\boldsymbol{x})$, arises from the intrinsic noise on the data and represents the minimum achievable value of the expected loss.

- If we model $h(\boldsymbol{x})$ using a parametric function $y(\boldsymbol{x}, \boldsymbol{w})$ governed by $\boldsymbol{w}$, then from a Bayesian perspective the uncertainty in our model is expressed through a posterior distribution over $\boldsymbol{w}$.

The Bias-Variance Decomposition

$$\mathbb{E}[L] = \int \big(y(\boldsymbol{x}) - h(\boldsymbol{x})\big)^2 p(\boldsymbol{x}) d\boldsymbol{x} + \iint (h(\boldsymbol{x}) - t)^2 p(\mathbf{x}, t) d\boldsymbol{x} dt$$

- A frequentist treatment involves making a point estimate of $\boldsymbol{w}$ based on the data set $\mathcal{D}$, and interpret the uncertainty of this estimate as follows.

- Suppose we had a large number of data sets each of size N and each drawn independently from $p(\boldsymbol{x}, t)$.

- For any given data set $\mathcal{D}$, we run our learning algorithm and obtain a prediction function $y(\boldsymbol{x}; \mathcal{D})$. Different data sets from the ensemble give different functions and values of the squared loss.

- The performance of the learning algorithm is then assessed by taking the average over this ensemble of data sets.

The Bias-Variance Decomposition

- For any given data set $\mathcal{D}$, we can run our learning algorithm and obtain a prediction function $y(\boldsymbol{x};\mathcal{D})$.

$$\{y(\boldsymbol{x};\mathcal{D}) - h(\boldsymbol{x})\}^2 = \{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2$$
$$= \{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{D}[y(\boldsymbol{x};D)]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2 +$$
$$2\{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{D}[y(\boldsymbol{x};\mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}$$

- Here $\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]$ is the average of our prediction function over all data sets.

- Take the expectation of this expression with respect to $\mathcal{D}$ and note that the final term will vanish, giving

$$\mathbb{E}_{\mathcal{D}}[\{y(\boldsymbol{x};\mathcal{D}) - h(x)\}^2] = \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(x)\}^2}_{(Bias)^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\}^2]}_{Variance}$$

- Recall that $h(\boldsymbol{x})$ is the desired regression function.

The Bias-Variance Decomposition

$$\mathbb{E}[L] = \int \big(y(\boldsymbol{x}) - h(\boldsymbol{x})\big)^2 p(\boldsymbol{x}) d\boldsymbol{x} + \iint (h(\boldsymbol{x}) - t)^2 p(\mathbf{x}, t) d\boldsymbol{x} dt$$

- So far, we have considered a single input value $\boldsymbol{x}$. If we substitute this expansion back into the expected squared loss function shown above, we obtain the following decomposition of the expected squared loss

Expected loss = $(\text{bias})^2$ + variance + noise

where

$$bias^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x}; \mathcal{D})] - h(\boldsymbol{x})\}^2 p(\boldsymbol{x}) d\boldsymbol{x}$$

$$variance = \int \mathbb{E}_{\mathcal{D}}[\{y(\boldsymbol{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x}; \mathcal{D})]\}^2] p(\boldsymbol{x}) d\boldsymbol{x}$$

$$noise = \iint (h(\boldsymbol{x}) - t)^2 p(\boldsymbol{x}, t) d\boldsymbol{x} dt$$

The Bias-Variance Decomposition

Expected loss = $(\text{bias})^2$ + variance + noise

$$bias^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2 p(\boldsymbol{x}) d\boldsymbol{x}$$

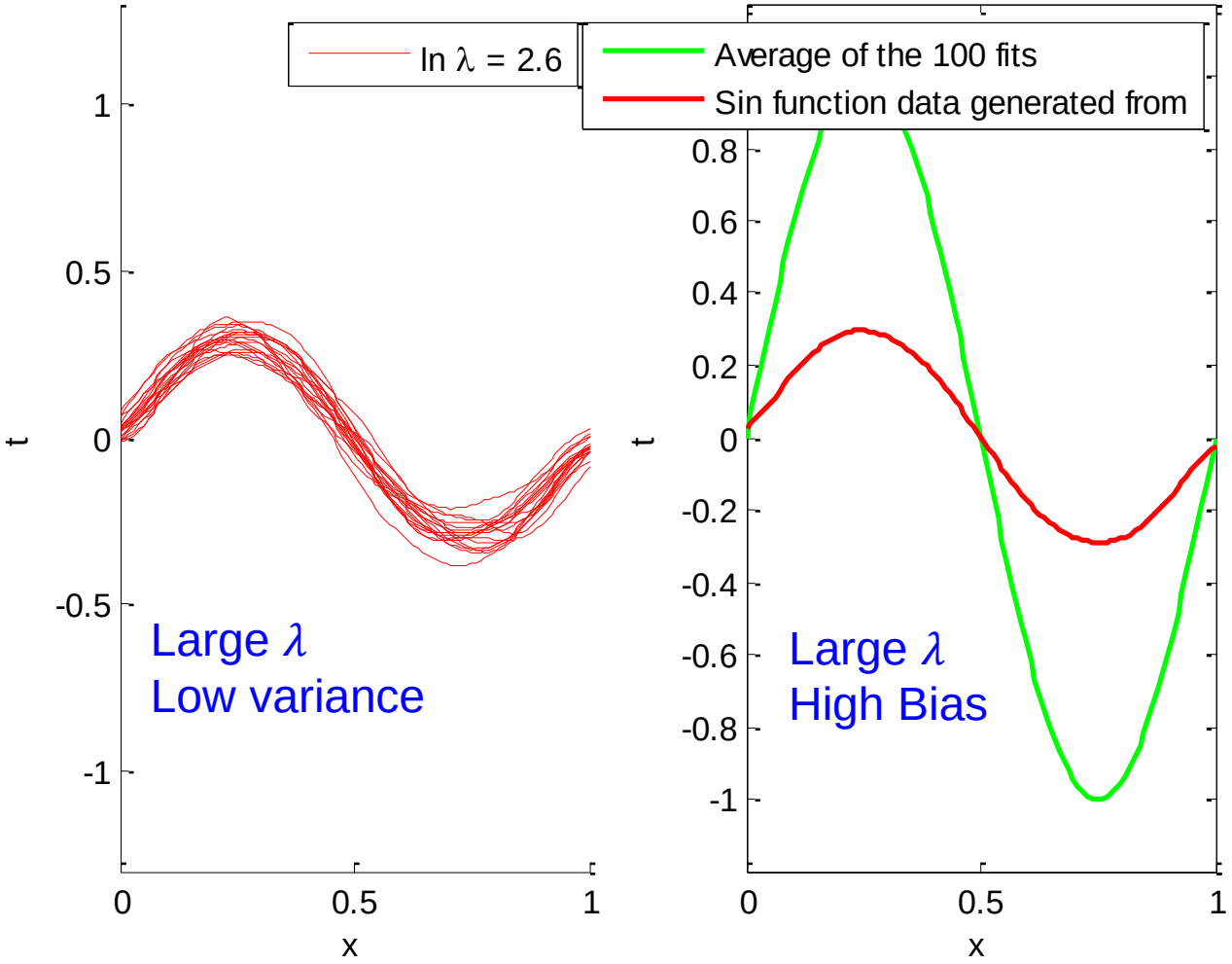
$$variance = \int \mathbb{E}_{\mathcal{D}}[\{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\}^2] p(\boldsymbol{x}) d\boldsymbol{x}$$

$$noise = \iint (h(\boldsymbol{x}) - t)^2 p(\boldsymbol{x}, t) d\boldsymbol{x} dt$$

- There is a tradeoff between bias and variance
  - flexible models have low bias and high variance
  - rigid models have high bias and low variance
- The bias-variance decomposition provides insights in model complexity but is of limited use since several data sets $\mathcal{D}$ are needed.

The Bias-Variance Decomposition

$L = 100$ *data sets, each with* $N = 25$ data points. 24
Gaussian basis functions, # of parameters $M = 25$

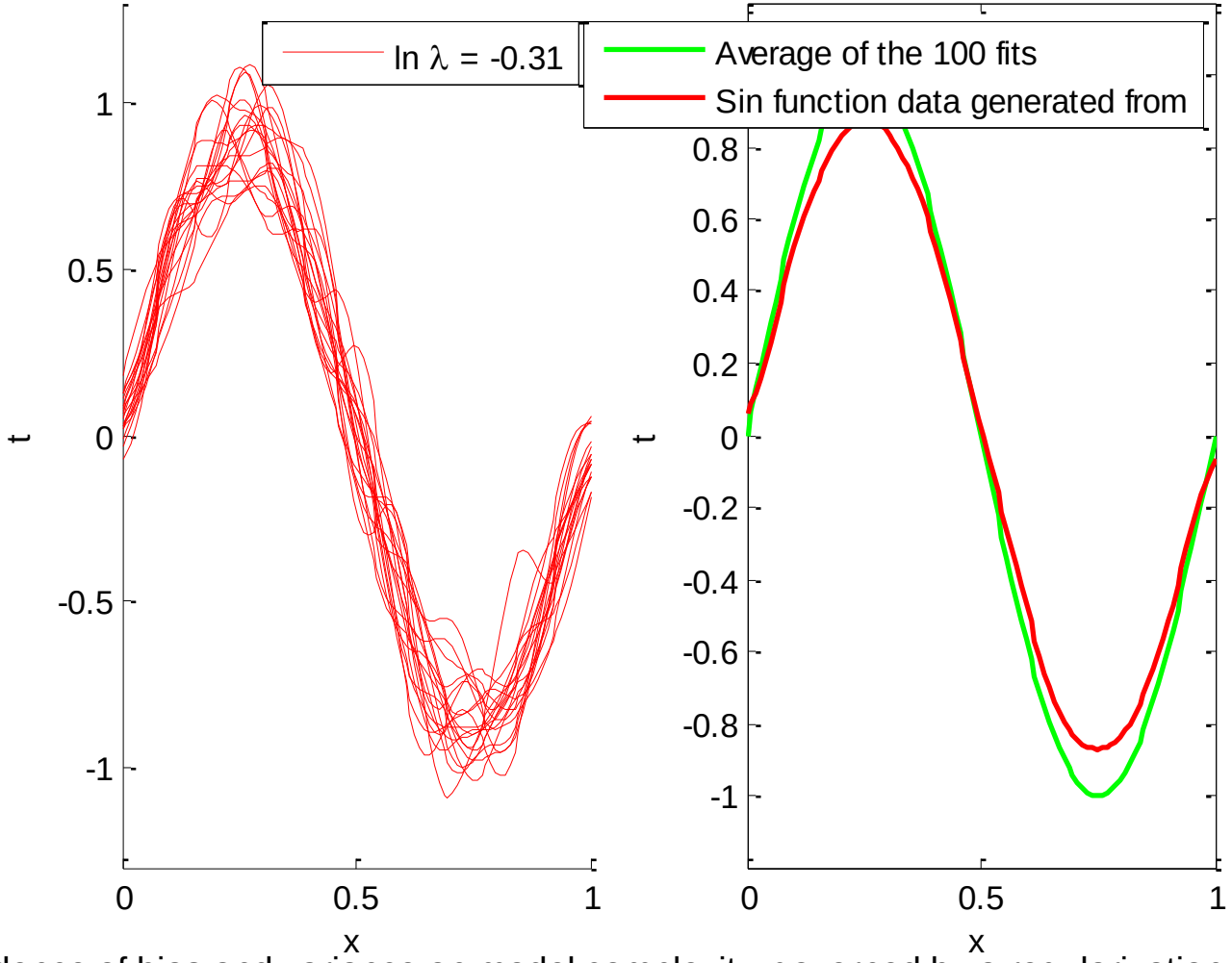


MatLab code

- Dependence of bias and variance on model complexity, governed by a regularization parameter λ, using the sinusoidal data set. On the left only 20 of the 100 fits are shown.

The Bias-Variance Decomposition

$L = 100$ *data sets, each with* $N = 25$ data points. 24
Gaussian basis functions, # of parameters $M = 25$



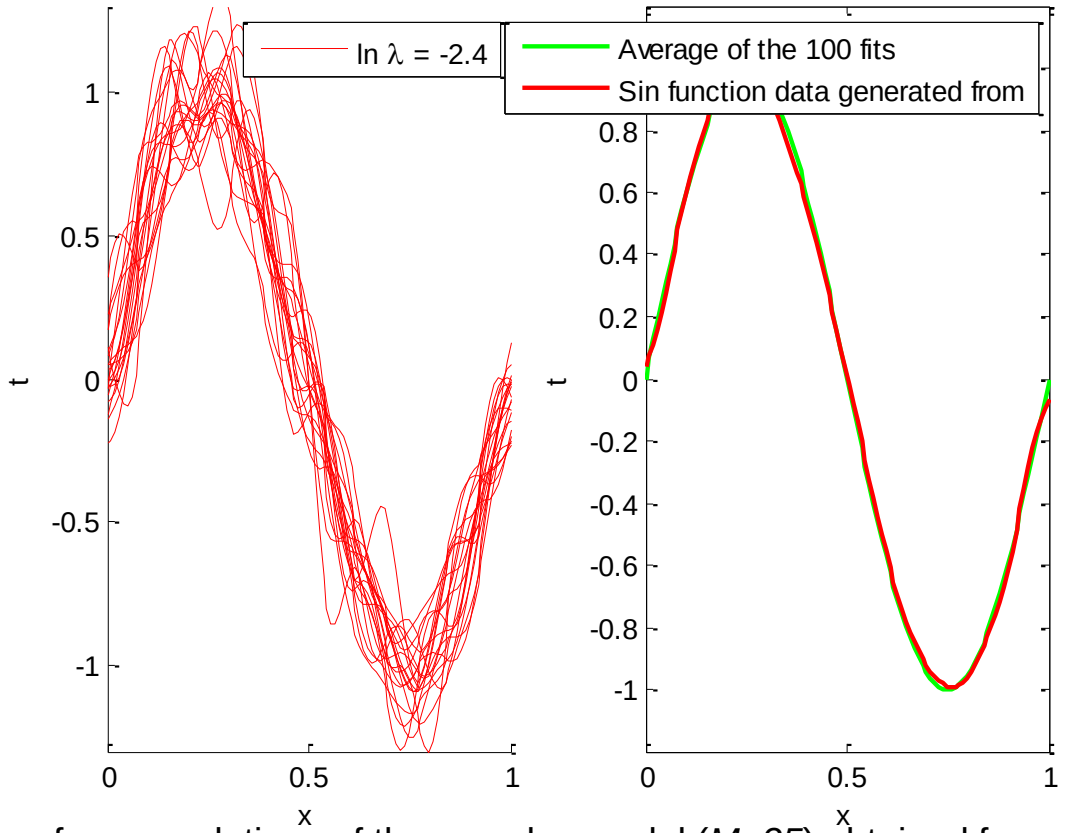
MatLab code

- Dependence of bias and variance on model complexity, governed by a regularization parameter λ, using the sinusoidal data set. On the left only 20 of the 100 fits are shown.

The Bias-Variance Decomposition

$L = 100$ *data sets, each with N = 25* data points. 24
Gaussian basis functions, # of parameters $M = 25$

Lowest λ
Large variance
Low Bias



MatLab code

- Note averaging of many solutions of the complex model (*M=25*) obtained from different data sets gives very good solution.
- Weighted averaging is also done in Bayesian approach but there it is with respect to the posterior distribution of parameters!

Trade-off quantities

- The average prediction over L data sets is estimated from

$$\bar{y}(x) = \frac{1}{L}\sum_{\ell=1}^{L} y^{(\ell)}(x)$$

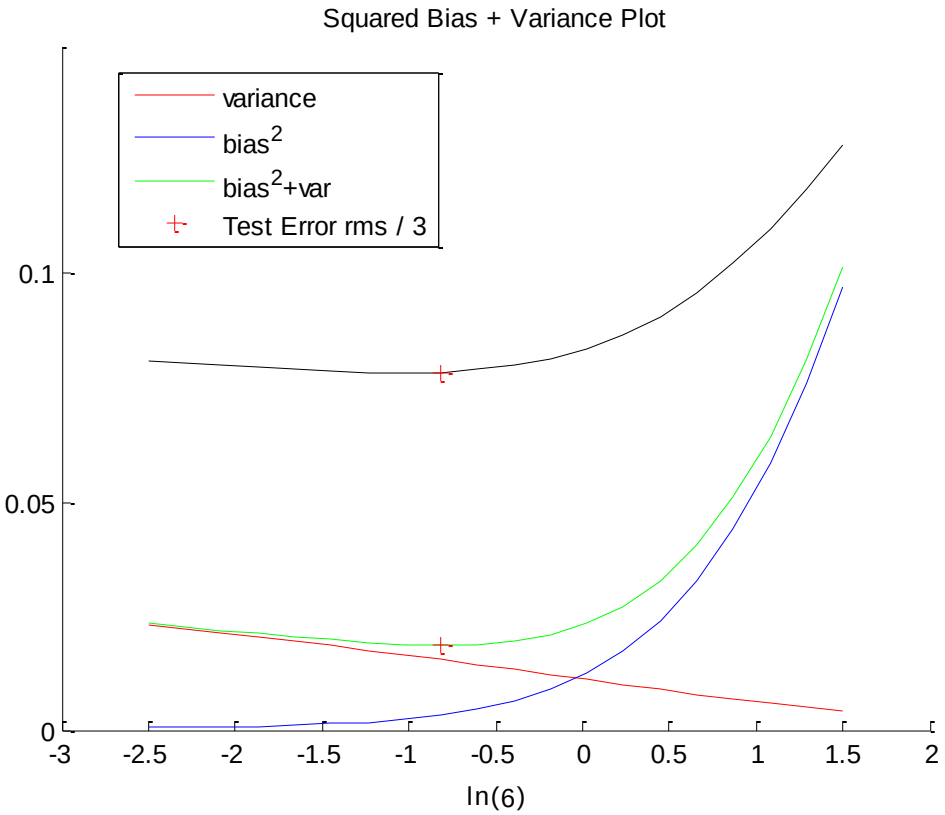
- The integrated squared bias and integrated variance are then given by

$$bias^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - h(\boldsymbol{x})\}^2 p(\boldsymbol{x})d\boldsymbol{x} \Rightarrow \quad (bias)^2 = \frac{1}{N}\sum_{n=1}^{N}\{\bar{y}(\boldsymbol{x}_n) - h(\boldsymbol{x}_n)\}^2$$

$$variance = \int \mathbb{E}_{\mathcal{D}}[\{y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\}^2]p(\boldsymbol{x})d\boldsymbol{x} \Rightarrow$$

$$variance = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{L}\sum_{\ell=1}^{L}\left\{y^{(\ell)}(\boldsymbol{x}_n) - \bar{y}(\boldsymbol{x}_n)\right\}^2$$

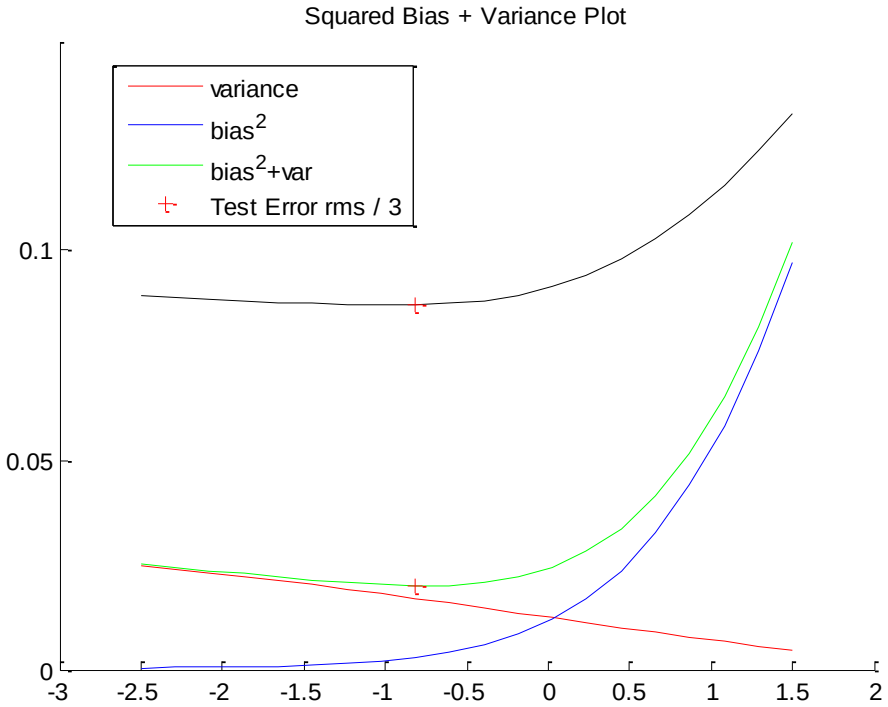
The Bias-Variance Decomposition



MatLab code

- Note that small regularization parameter λ allows the model to be finely tuned to the noise in each individual data set leading to small bias but large variance.
- Conversely, large λ forces all $\boldsymbol{w}$'s to go towards zero leading to large bias but small variance.

The Bias-Variance Decomposition



MatLab code

- Plot of squared bias and variance together with their sum. Also shown is the test set rms squared error for a test data set size of 1000 points.
- The minimum value of (bias)^2 + variance occurs around $\ln \lambda = -0.31$, which is near to the value that gives the minimum error on the test data.

MAP and Regularized Least Squares

- Now let us take a step towards a Bayesian approach and introduce a prior distribution over the coefficients $\boldsymbol{w}$. Consider a Gaussian distribution

$$p(\boldsymbol{w}\,|\,\alpha) = \mathcal{N}(\boldsymbol{w}\,|\,0, \alpha^{-1}I) = \left(\frac{\alpha}{2\pi}\right)^{M/2} \exp\left\{-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\right\}$$

where α is the precision of the distribution, and M is the total number of elements in the vector $\boldsymbol{w}$.*

- Using Bayes' theorem:

$$p(\boldsymbol{w}\,|\,\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) \propto p(\boldsymbol{t}\,|\,\boldsymbol{x}, \boldsymbol{w}, \beta)\, p(\boldsymbol{w}\,|\,\alpha)$$

* Note that one should not penalize the bias term $\boldsymbol{w}_0$ as it does not contribute to overfitting. We will discuss on a follow up lecture how to address this by treating the bias term separately by working with "centered data" (both responses $\boldsymbol{t}_c = \boldsymbol{t} - \bar{\boldsymbol{t}} = \boldsymbol{t} - \mathbf{1}_N \bar{t}, \bar{t} \equiv \sum_{i=1}^{N} t_i/N$, and input variables s.t. $y(\boldsymbol{x}_n, \boldsymbol{w}) = \boldsymbol{\phi}_c(\boldsymbol{x}_n)^T\boldsymbol{w}$, $\boldsymbol{\Phi}_c = (\boldsymbol{\varphi}_{1c} \quad \boldsymbol{\varphi}_{2c} \quad \cdots \quad \boldsymbol{\varphi}_{Mc})$, $\boldsymbol{\varphi}_{ic} = \boldsymbol{\varphi}_i - \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\varphi}_i(\boldsymbol{x}_n) = 0, i = 1, .. M-1)$. Here $\boldsymbol{\phi}^T$ are rows and $\boldsymbol{\varphi}$ columns of the $N \times M$ design matrix $\boldsymbol{\Phi}$.

MAP Estimate

- We can determine $\boldsymbol{w}$ (point estimate) by finding the most probable value of $\boldsymbol{w}$ given the data, i.e. maximizing the posterior.

- This technique is called *maximum posterior (MAP).*

- The maximum of the posterior is given by the minimum of

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n,\boldsymbol{w})-t_n\}^2+\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}$$

- Note that the MAP point estimate is equivalent to regularized sum of squares error function with regularization parameter

$$\lambda=\frac{\alpha}{\beta}=\frac{precision\ of\ \ prior}{precision\ in\ the\ data}$$

Posterior Distribution

$$p(\boldsymbol{w}\,|\,\alpha)=\left(\frac{\alpha}{2\pi}\right)^{M/2}\exp\left\{-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\right\},\; p(\boldsymbol{t}\,|\,\boldsymbol{x},\boldsymbol{w},\beta)\propto\exp\left\{-\frac{\beta}{2}\sum_{n=1}^{N}\left(\boldsymbol{\phi}(x_n)^T\boldsymbol{w}-t_n\right)^2\right\}\Rightarrow$$

$$p(\boldsymbol{t}\,|\,\boldsymbol{x},\boldsymbol{w},\beta)\propto\exp\left\{-\frac{1}{2}\boldsymbol{w}^T\sum_{n=1}^{N}\beta\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T\boldsymbol{w}+\beta\boldsymbol{w}^T\sum_{n=1}^{N}t_n\boldsymbol{\phi}(x_n)\right\}$$

Quadratic in $\boldsymbol{w}$

- We now have the product of two Gaussians and the posterior is easily computed as:

Completing the square $-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu})^T\boldsymbol{S}_N^{-1}(\boldsymbol{w}-\boldsymbol{\mu})$

$$p(\boldsymbol{w}\,|\,\boldsymbol{x},\boldsymbol{t},\alpha,\beta)\propto$$

$$\exp\left(-\frac{1}{2}\boldsymbol{w}^T\alpha\boldsymbol{I}_{M\times M}\boldsymbol{w}-\frac{1}{2}\boldsymbol{w}^T\sum_{n=1}^{N}\beta\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T\boldsymbol{w}+\beta\boldsymbol{w}^T\sum_{n=1}^{N}t_n\boldsymbol{\phi}(x_n)\right)$$

$$\propto\mathcal{N}\left(\boldsymbol{w}\,|\,\beta\boldsymbol{S}_N\sum_{n=1}^{N}t_n\boldsymbol{\phi}(x_n),\boldsymbol{S}_N\right),\;\boldsymbol{S}_N^{-1}=\alpha\boldsymbol{I}+\sum_{n=1}^{N}\beta\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T$$

E.g. for polynomial regression: $\boldsymbol{\phi}(x_n)=\begin{Bmatrix}1\\x_n\\x_n^2\\:\\x_n^{M-1}\end{Bmatrix}$, $\boldsymbol{\phi}(x)^T=\begin{Bmatrix}1 & x & x^2 & .. & x^{M-1}\end{Bmatrix}$, $\boldsymbol{I}=$ *unit matrix* $M\times M$

Posterior Distribution

- For a prior

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid 0, \alpha^{-1}\boldsymbol{I})$$

the corresponding posterior distribution over $\boldsymbol{w}$ is then given by

$$p(\boldsymbol{w}|\boldsymbol{t}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N, \boldsymbol{S}_N)$$

$$\boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}$$

$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$$

General Gaussian as a Prior $p(w)$

- Assume additive Gaussian noise with known precision β. The likelihood function $p(\boldsymbol{t}|\boldsymbol{w})$ is the exponential of a quadratic function of $\boldsymbol{w}$

$$p(\boldsymbol{t}\mid \boldsymbol{X},\boldsymbol{w},\beta)=\prod_{n=1}^{N}\mathcal{N}\left(t_n\mid \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n),\beta^{-1}\right)=\left(\frac{\beta}{2\pi}\right)^{N/2}\exp\left(-\frac{\beta}{2}\sum_{n=1}^{N}\left\{t_n-\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right\}^2\right)$$

and its conjugate prior is Gaussian:

$$p(\boldsymbol{w})=\mathcal{N}(\boldsymbol{w}\mid \boldsymbol{m}_0,\boldsymbol{S}_0)$$

- Combining this with the likelihood and using results for marginal and conditional Gaussian distributions, gives the posterior

$$p(\boldsymbol{w}\mid \boldsymbol{t})=\mathcal{N}(\boldsymbol{w}\mid \boldsymbol{m}_N,\boldsymbol{S}_N)$$

where

$$\boldsymbol{m}_N=\boldsymbol{S}_N\left(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0+\beta\boldsymbol{\Phi}^T\boldsymbol{t}\right)$$

$$\boldsymbol{S}_N^{-1}=\boldsymbol{S}_0^{-1}+\beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}$$

Posterior Distribution: Derivation

$$p(\boldsymbol{w}\,|\,\boldsymbol{m}_0,\boldsymbol{S}_0)\propto\exp\left\{-\frac{1}{2}\left(\boldsymbol{w}-\boldsymbol{m}_0\right)^T\boldsymbol{S}_0^{-1}\left(\boldsymbol{w}-\boldsymbol{m}_0\right)\right\},$$

$$p(\boldsymbol{t}\,|\,\boldsymbol{x},\boldsymbol{w},\beta)\propto\exp\left\{-\frac{\beta}{2}\sum_{n=1}^{N}\left(\boldsymbol{\phi}(x_n)^T\boldsymbol{w}-t_n\right)^2\right\}\Rightarrow$$

$$p(\boldsymbol{t}\,|\,\boldsymbol{x},\boldsymbol{w},\beta)\propto\exp\left\{-\frac{1}{2}\boldsymbol{w}^T\sum_{n=1}^{N}\beta\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{w}+\beta\sum_{n=1}^{N}t_n\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{w}\right\}$$

- We now have the product of two Gaussians and the posterior is easily computed as:

Complete the square in $\boldsymbol{w}$

$$p(\boldsymbol{w}\,|\,\boldsymbol{x},\boldsymbol{t},\boldsymbol{S}_0,\beta)\propto$$

$$\exp\left(-\frac{1}{2}\boldsymbol{w}^T\boldsymbol{S}_0^{-1}\boldsymbol{w}-\boldsymbol{w}^T\boldsymbol{S}_0^{-1}\boldsymbol{m}_0-\frac{1}{2}\boldsymbol{w}^T\sum_{n=1}^{N}\beta\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{w}+\beta\boldsymbol{w}^T\sum_{n=1}^{N}t_n\boldsymbol{\phi}(\boldsymbol{x}_n)\right)$$

$$\propto\mathcal{N}\left(\boldsymbol{w}\,|\,\underbrace{\boldsymbol{S}_N\left(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0+\beta\Phi^T\boldsymbol{t}\right)}_{\boldsymbol{m}_N},\boldsymbol{S}_N\right),\ \boldsymbol{S}_N^{-1}=\boldsymbol{S}_0^{-1}+\sum_{n=1}^{N}\beta\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T=\boldsymbol{S}_0^{-1}+\beta\Phi^T\Phi$$

Sequential Posterior Calculation

- Note that because the posterior distribution is Gaussian, its posterior mode coincides with its mean.

$$p(\boldsymbol{w}\,|\,\boldsymbol{t}) = \mathcal{N}(\boldsymbol{w}\,|\,\boldsymbol{m}_N, \boldsymbol{S}_N)$$

$$\boldsymbol{m}_N = \boldsymbol{S}_N\left(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta\boldsymbol{\Phi}^T\boldsymbol{t}\right)$$

$$\boldsymbol{S}_N^{-1} = \boldsymbol{S}_0^{-1} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}$$

$$\boldsymbol{w}_{MAP} = \boldsymbol{m}_N$$

- The above expressions for the posterior mean and variance can also be written for a sequential calculation (we already have observed N data points and now considering an additional data point $(\boldsymbol{x}_{N+1}, t_{N+1})$). In this case, we have:

$$p(\boldsymbol{w}\,|\,t_{N+1}, \boldsymbol{x}_{N+1}, \boldsymbol{m}_N, \boldsymbol{S}_N) = \mathcal{N}(\boldsymbol{w}\,|\,\boldsymbol{m}_{N+1}, \boldsymbol{S}_{N+1})$$

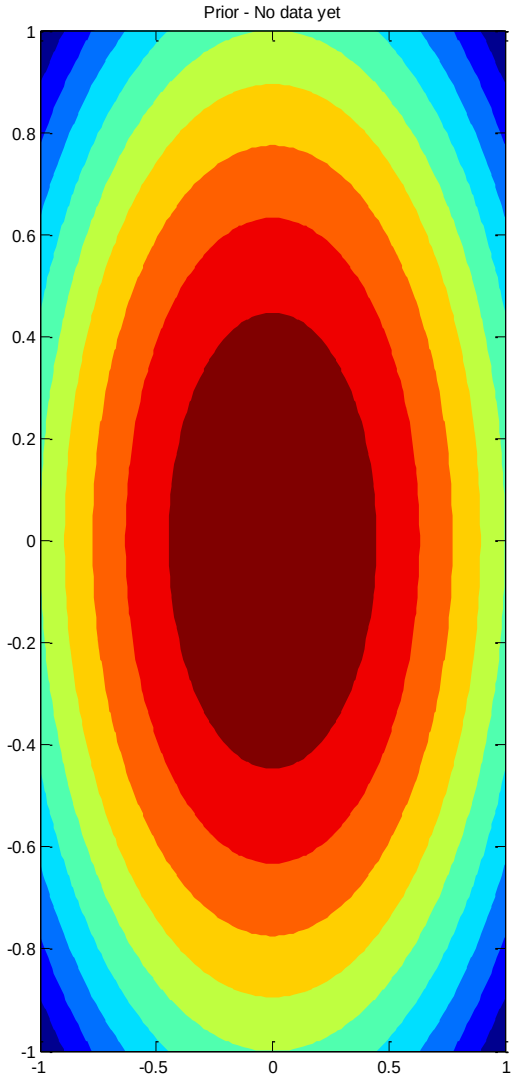
$$\boldsymbol{m}_{N+1} = \boldsymbol{S}_{N+1}\left(\boldsymbol{S}_N^{-1}\boldsymbol{m}_N + \beta\boldsymbol{\phi}_{n+1}t_{n+1}\right)$$

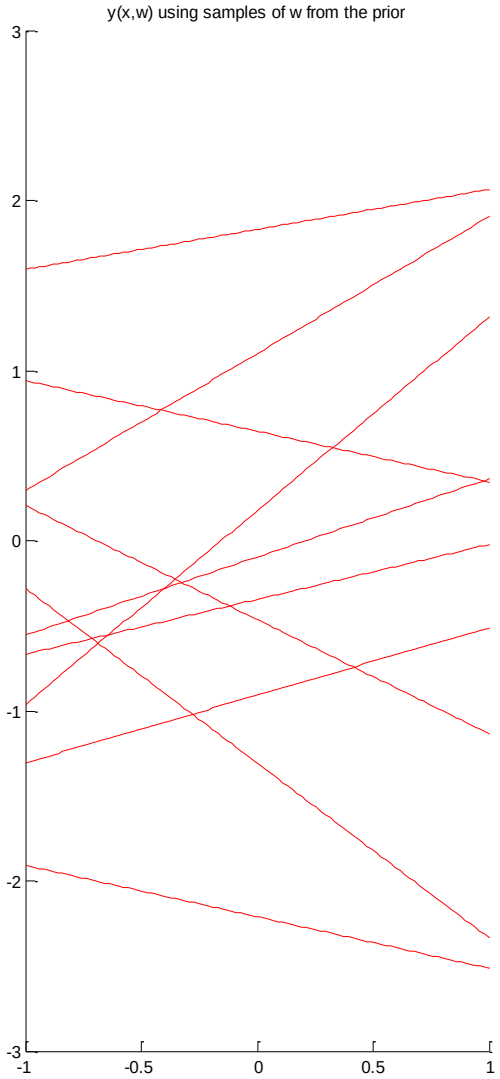
$$\boldsymbol{S}_{N+1}^{-1} = \boldsymbol{S}_N^{-1} + \beta\boldsymbol{\phi}_{n+1}\boldsymbol{\phi}_{n+1}^T$$

Bayesian Regression: Example

- We generate synthetic data from the function $f(x, a) = a_0 + a_1 x$ with parameter values $a_0 = -0.3$ and $a_1 = 0.5$ by first choosing values of x_n from the uniform distribution $\mathcal{U}(x|-1, 1)$, then evaluating $f(x_n, a)$, and finally adding Gaussian noise with standard deviation of 0.2 to obtain the target values t_n.
- We assume $\beta = (1/0.2)^2 = 25$ and $\alpha = 2.0$.
- We perform Bayesian inference sequentially – one point at a time – so the posterior at each level becomes the new prior.
- We show results after $1, 2$ and 22 points have been collected.
- The results include the likelihood contours (for 1 point), the posterior and samples of the regression function from the posterior.

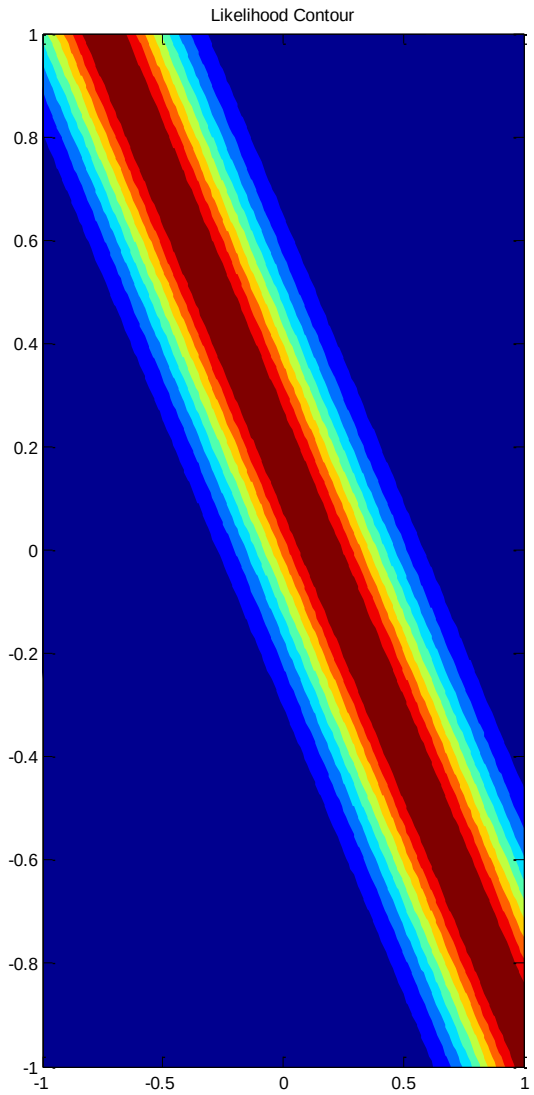
Bayesian Regression: Example

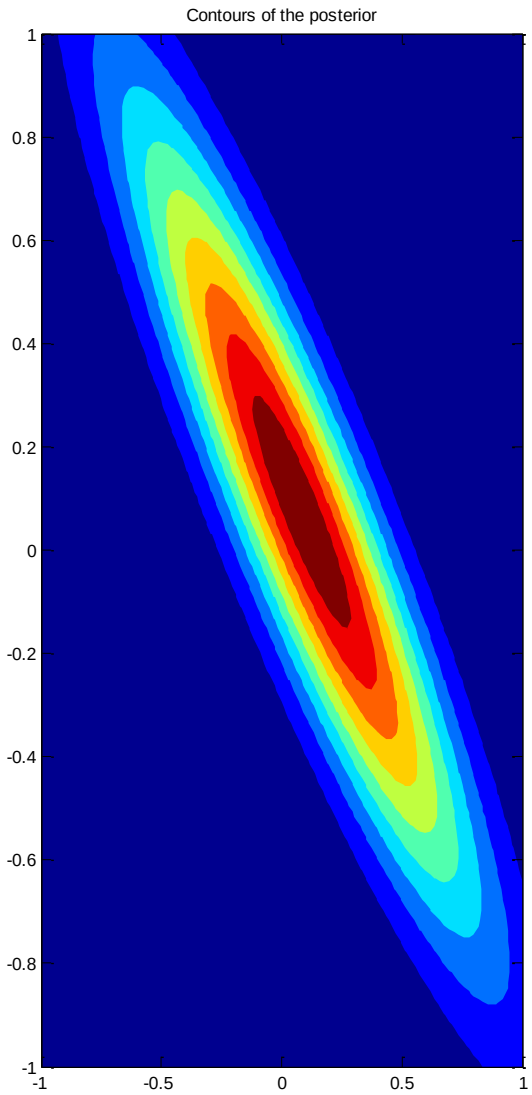


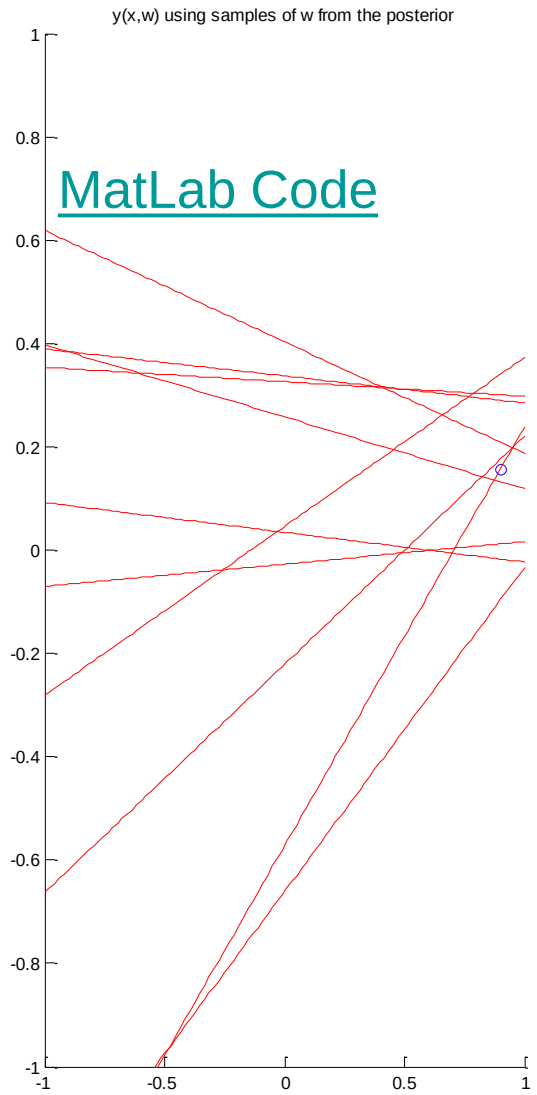


MatLab Code

Example: One Data Point Collected

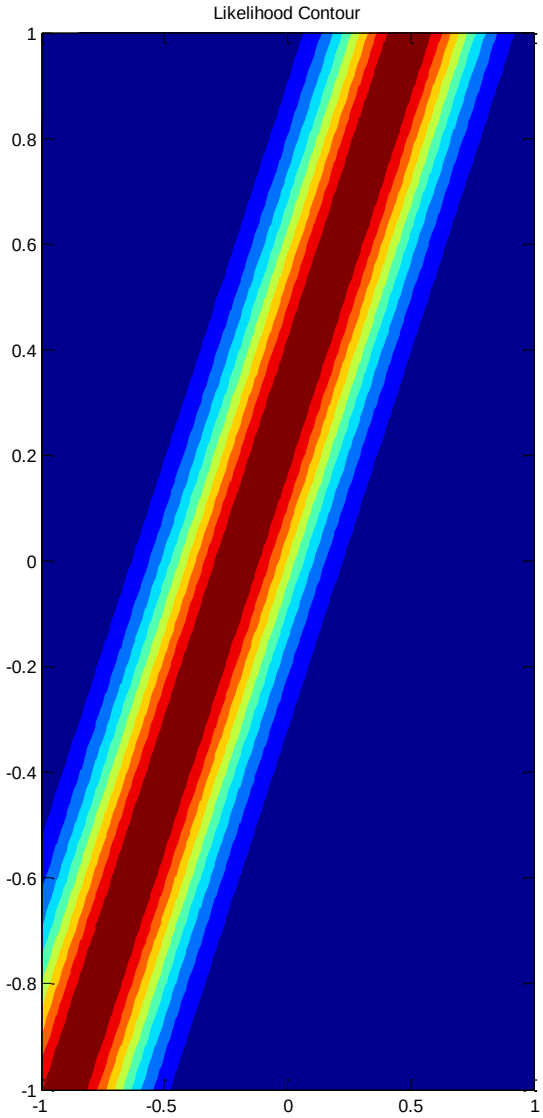


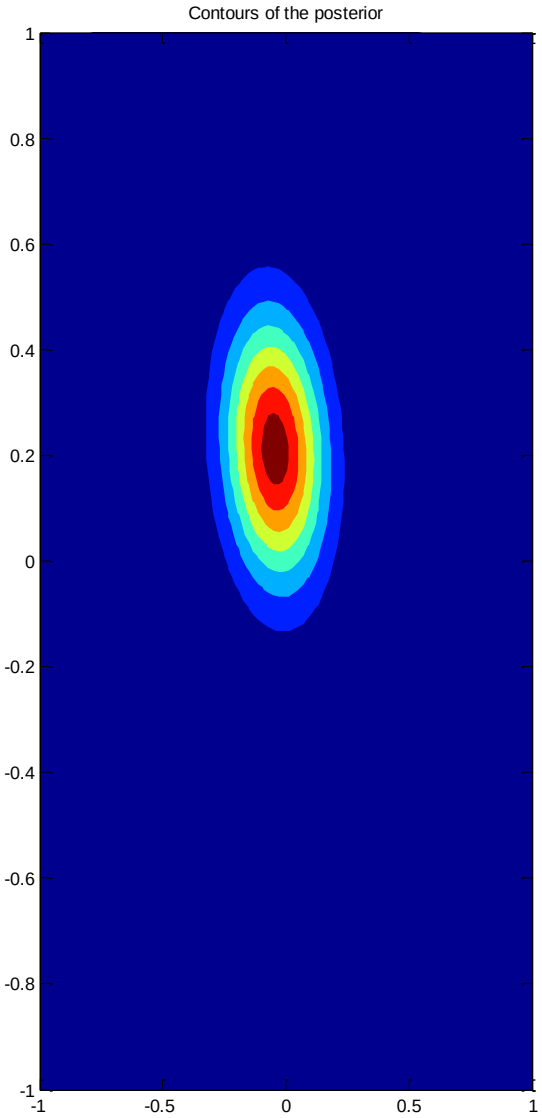


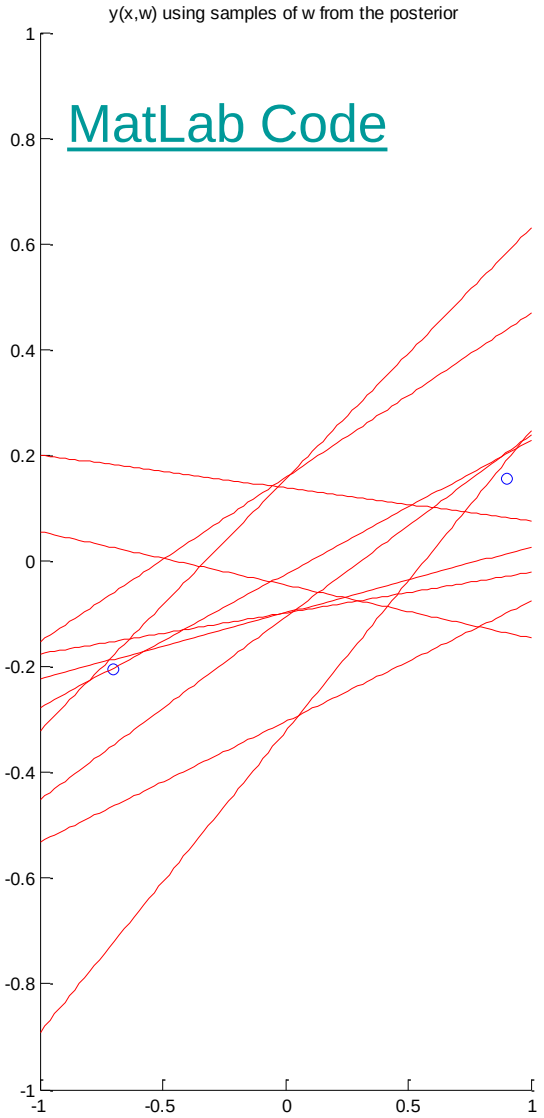


Note that the regression lines pass close to the data point (shown with a circle)

Example: 2nd Data Point Collected



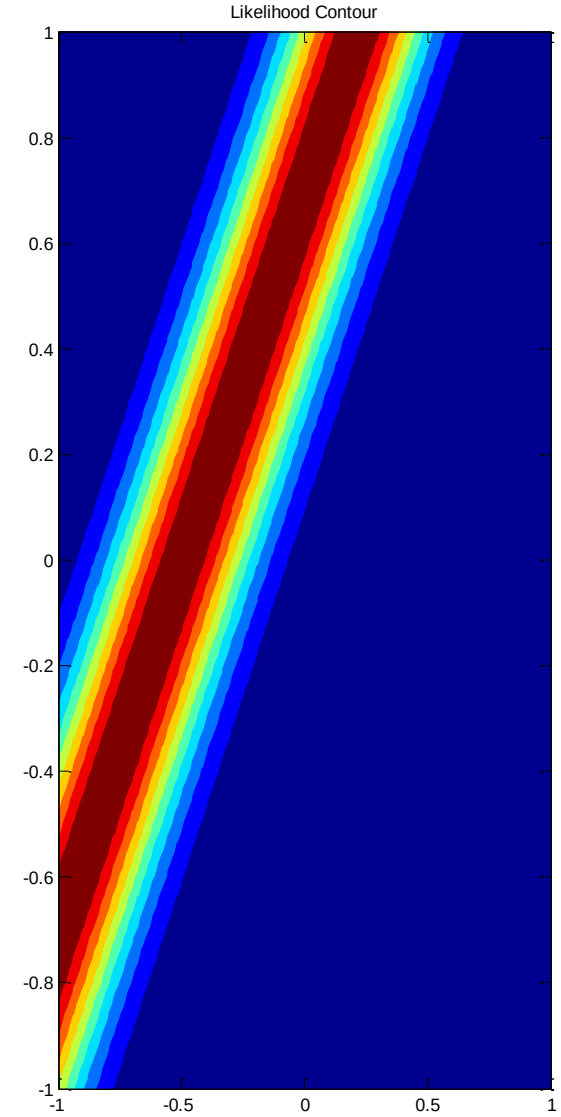


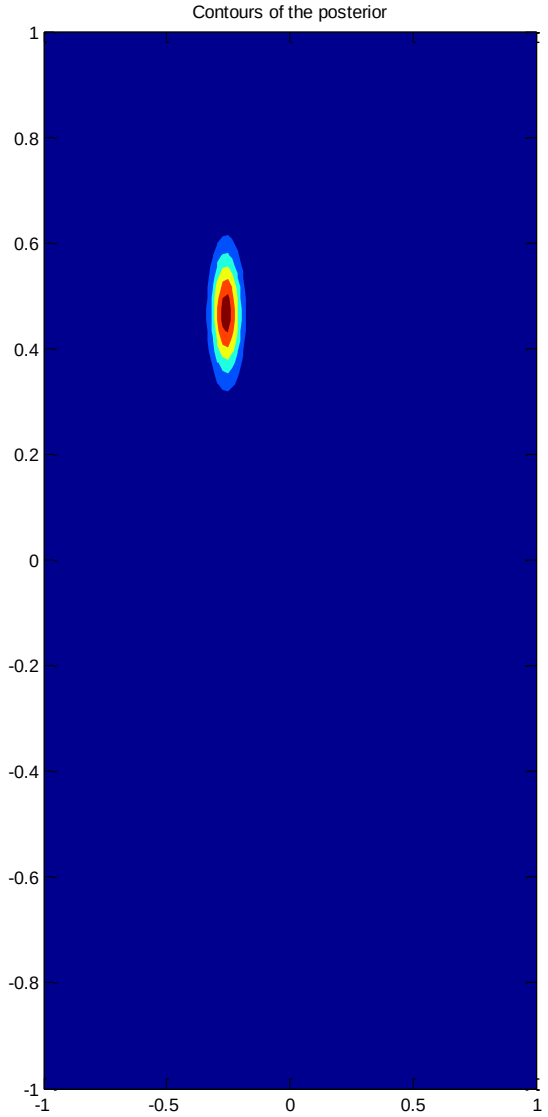


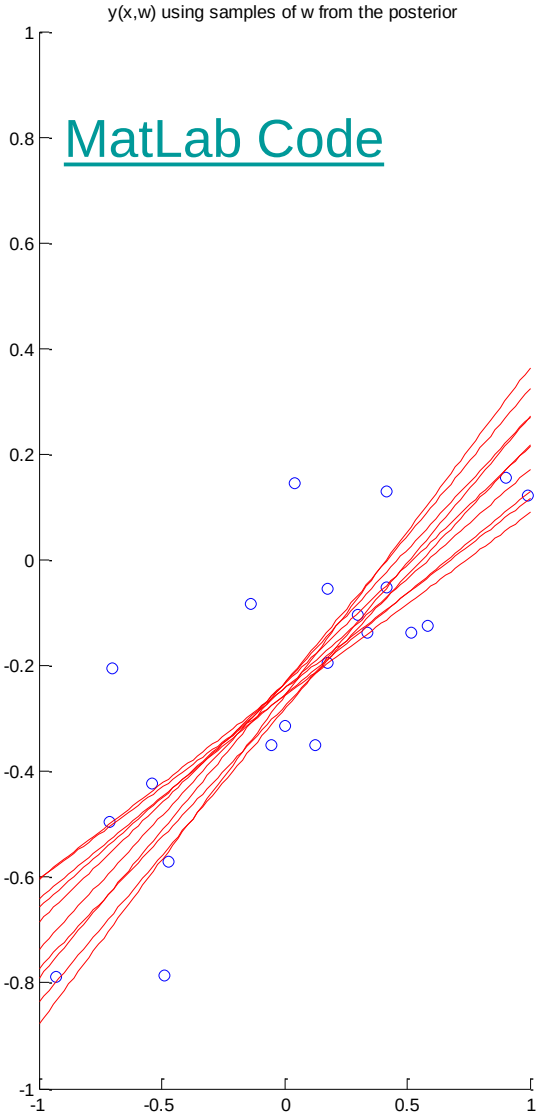
MatLab Code

Note that the regression lines now pass close to both data points

Example: 22 Data Points Collected



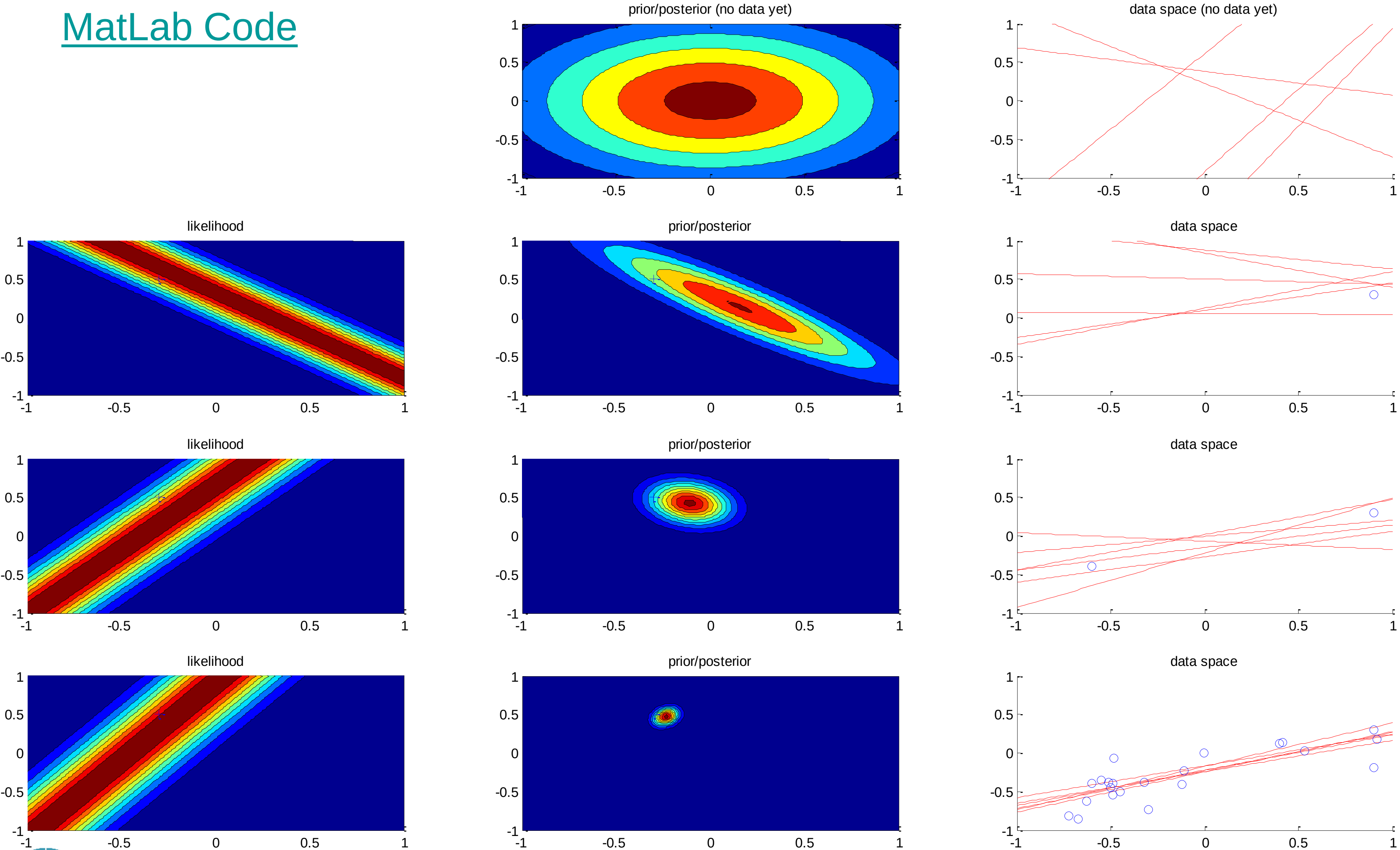




Note that the regression lines after 22 data points have been collected

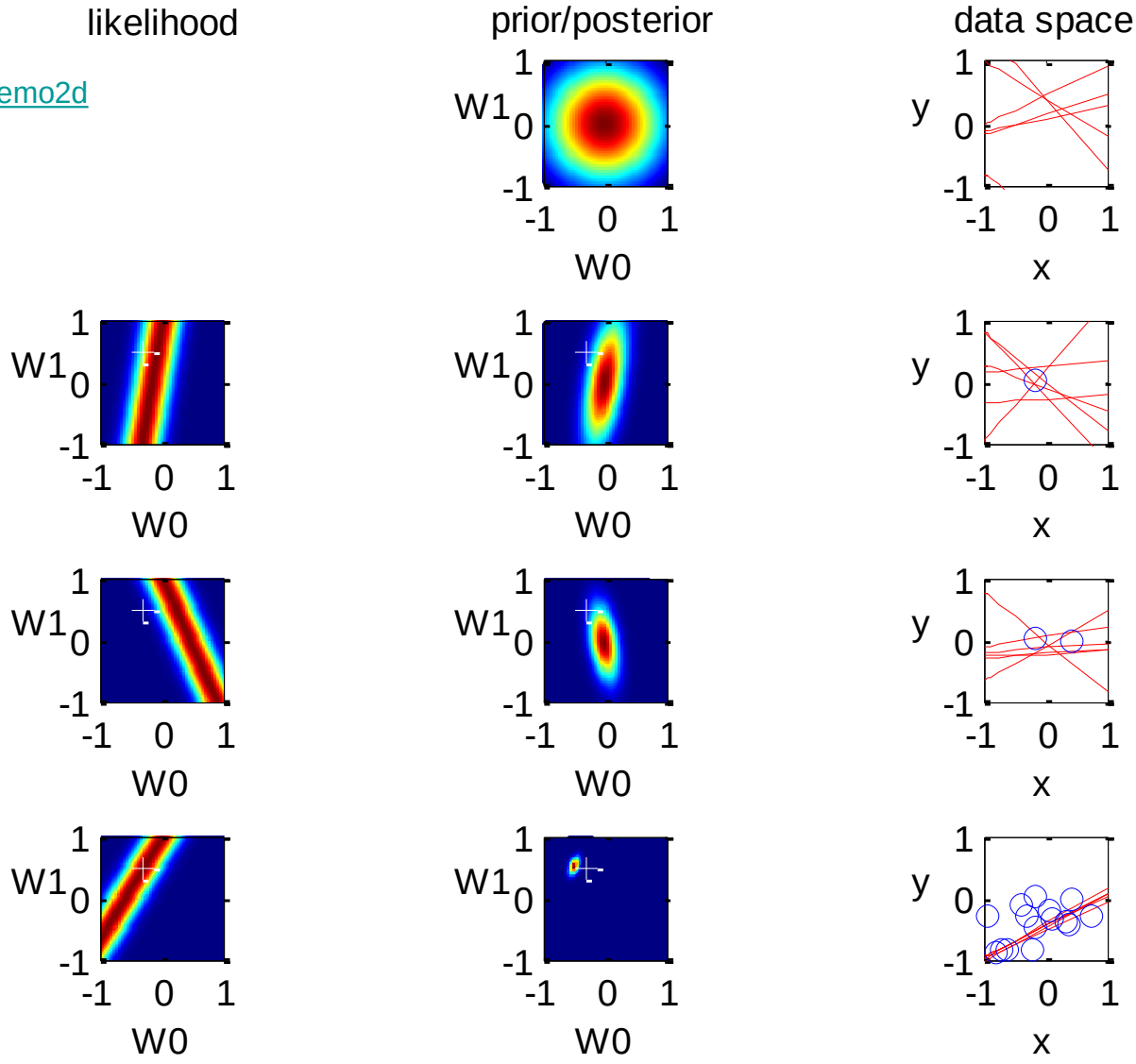
Summary of Results

MatLab Code



Summary of Results

Run bayesLinRegDemo2d from PMTK3



Predictive Distribution

- In a full Bayesian treatment, we want to compute the predictive distribution, i.e. given the training data $\boldsymbol{x}$ and $\boldsymbol{t}$ and a new test point x, we want the distribution:

$$p(t \mid x, \boldsymbol{x}, \boldsymbol{t}) = \int p(t \mid x, \boldsymbol{w})\, p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{t}) d\boldsymbol{w}, \quad where$$

$$p(t \mid x, \boldsymbol{w}, \beta) = \mathcal{N}(t \mid y(x, \boldsymbol{w}), \beta^{-1}) = \mathcal{N}(t \mid \boldsymbol{\phi}(x)^T \boldsymbol{w}, \beta^{-1}) \; and$$

$$p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) \propto \mathcal{N}\left(\boldsymbol{w} \mid \beta \boldsymbol{S}_N \sum_{n=1}^{N} t_n \boldsymbol{\phi}(x_n), \boldsymbol{S}_N \right), \; \boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \sum_{n=1}^{N} \beta \boldsymbol{\phi}(x_n) \boldsymbol{\phi}(x_n)^T$$

- To integrate $\boldsymbol{w}$ out as shown on the predictive distribution expression, we use a fundamental result for linear Gaussian models.

Appendix: Linear Gaussian Models

$$p(\boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}, \Lambda^{-1}\right)$$

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, \boldsymbol{L}^{-1}\right)$$

- For the above linear Gaussian model, the following very useful results hold:

$$p(\boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{L}^{-1} + \boldsymbol{A}\Lambda^{-1}\boldsymbol{A}^T\right)$$

$$p(\boldsymbol{x} \mid \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{x} \mid \left(\Lambda + \boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A}\right)^{-1}\left(\Lambda\boldsymbol{\mu} + \boldsymbol{A}^T\boldsymbol{L}(\boldsymbol{y} - \boldsymbol{b})\right), \left(\Lambda + \boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A}\right)^{-1}\right)$$

Predictive Distribution

$$p(\mathbf{x}) = \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}, \Lambda^{-1}\right)$$

$$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}\right)$$

$$p(\mathbf{w} \mid \mathbf{x}, \mathbf{t}, \alpha, \beta) = \mathcal{N}\left(\mathbf{w} \mid \beta \mathbf{S}_N \sum_{n=1}^{N} t_n \boldsymbol{\phi}(x_n), \mathbf{S}_N\right)$$

$$p(t \mid x, \mathbf{w}, \beta) = \mathcal{N}(t \mid y(x, \mathbf{w}), \beta^{-1})$$

- Thus for our problem:

$$\mathbf{x} \leftarrow \mathbf{w}, \boldsymbol{\mu} = \beta \mathbf{S}_N \sum_{n=1}^{N} t_n \boldsymbol{\phi}(x_n), \Lambda^{-1} = \mathbf{S}_N$$

$$\mathbf{y} \leftarrow t, \mathbf{A} = \boldsymbol{\phi}(x)^T, \mathbf{b} = 0, \mathbf{L}^{-1} = \beta^{-1}$$

- The predictive distribution now takes the form:

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\Lambda^{-1}\mathbf{A}^T\right) \Rightarrow$$

$$p(t) = \mathcal{N}\left(t \mid \boldsymbol{\phi}(x)^T \beta \mathbf{S}_N \sum_{n=1}^{N} t_n \boldsymbol{\phi}(x_n), \beta^{-1} + \boldsymbol{\phi}(x)^T \mathbf{S}_N \boldsymbol{\phi}(x)\right)$$

Predictive Distribution

- In a full Bayesian treatment, we want to compute the predictive distribution, i.e. given the training data $\boldsymbol{x}$ and $\boldsymbol{t}$ and a new test point x, we want the distribution:

$$p(t \mid x, \mathbf{x}, \mathbf{t}) = \int p(t \mid x, \mathbf{w})\, p(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) d\mathbf{w}, \quad p(t \mid x, \mathbf{w}, \beta) = \mathcal{N}(t \mid y(x, \mathbf{w}), \beta^{-1}) \Rightarrow$$

$$p(t \mid x, \mathbf{x}, \mathbf{t}) = \mathcal{N}(t \mid m(x), s^2(x))$$

where the mean and variance were shown in the earlier slide to be:

$$m(x) = \beta \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) t_n$$

$$s^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \boldsymbol{\phi}(x)$$

$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) \boldsymbol{\phi}(x)^T$$

Predictive Distribution

- The notation used here is as follows:

$$p(t \mid x, \boldsymbol{x}, \boldsymbol{t}) = \mathcal{N}(t \mid m(x), s^2(x))$$

$$m(x) = \beta \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) t_n$$

$$s^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \boldsymbol{\phi}(x)$$

$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) \boldsymbol{\phi}(x_n)^T$$

Predictive mean and variance are functions of x.

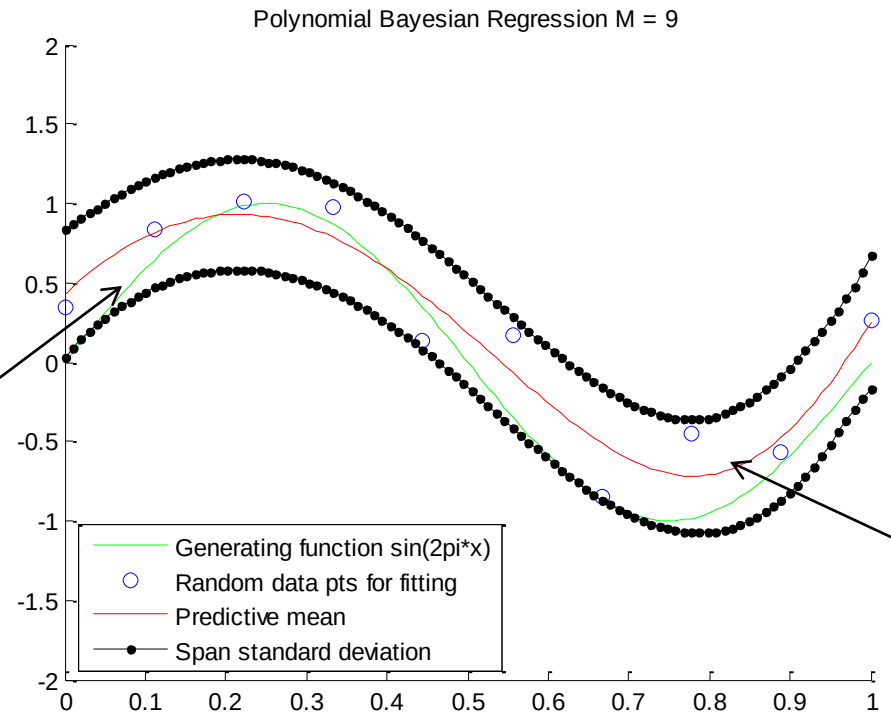
$$Eg.\ for\ polynomial\ regression: \boldsymbol{\phi}(x_n) = \begin{Bmatrix} 1 \\ x_n \\ x_n^2 \\ \vdots \\ x_n^{M-1} \end{Bmatrix},$$

$$\boldsymbol{\phi}(x)^T = \begin{Bmatrix} 1 & x & x^2 & .. & x^{M-1} \end{Bmatrix}, \boldsymbol{I} = unit\ matrix\ M \times M$$

Predictive Distribution

- The predictive distribution using an $M = 9$ polynomial, with fixed parameters $\alpha = 5 \times 10^{-3}$ *and* $\beta = 11.1$ (corresponding to a known noise variance).

- The red curve denotes the mean of the predictive distribution and the red region corresponds to ± standard deviation around the mean.



Generating function

Mean of the predictive distribution

MatLab Code

Predictive Distribution

- In a full Bayesian treatment, we want to compute the predictive distribution, i.e. given the training data $\boldsymbol{x}$ and $\boldsymbol{t}$ and a new test point x, we want the distribution:

$$p(t \mid x, \mathbf{x}, \mathbf{t}) = \int p(t \mid x, \boldsymbol{w})\, p(\boldsymbol{w} \mid \mathbf{x}, \mathbf{t})\, d\boldsymbol{w}, \quad p(t \mid x, \boldsymbol{w}, \beta) = \mathcal{N}(t \mid y(x, \boldsymbol{w}), \beta^{-1}) \Rightarrow$$

$$p(t \mid x, \mathbf{x}, \mathbf{t}) = \mathcal{N}(t \mid m(x), \sigma_N^2(x))$$

where the mean and variance are given by

$$m(x) = \beta \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) t_n = \boldsymbol{\phi}(x)^T \boldsymbol{m}_N, \quad \boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \mathbf{t}$$

$$\sigma_N^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \boldsymbol{\phi}(x) \quad (\textit{uncertainty in the data} + \textit{uncertainty in } \boldsymbol{w})$$

$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) \boldsymbol{\phi}(x_n)^T = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$$

- Note that: $\sigma_N^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \boldsymbol{\phi}(x) \overset{N \to \infty}{\rightarrow} \beta^{-1}$ and $\sigma_{N+1}^2(x) \leq \sigma_N^2(x)$

Predictive Distribution

- It is easy to show:

$$\sigma_N^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \boldsymbol{\phi}(x) \overset{N\to\infty}{\to} \beta^{-1} \text{ and } \sigma_{N+1}^2(x) \le \sigma_N^2(x)$$

- Note: $\boldsymbol{S}_{N+1}^{-1} = \alpha \boldsymbol{I} + \beta \sum_{n=1}^{N+1} \boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T = \boldsymbol{S}_N^{-1} + \beta \boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T$

and the identity:

$$\left(\boldsymbol{M} + \boldsymbol{v}\boldsymbol{v}^T\right)^{-1} = \boldsymbol{M}^{-1} - \frac{\left(\boldsymbol{M}^{-1}\boldsymbol{v}\right)\left(\boldsymbol{v}^T\boldsymbol{M}^{-1}\right)}{1 + \boldsymbol{v}^T\boldsymbol{M}^{-1}\boldsymbol{v}}$$

- Using these results, we can write:

$$\sigma_{N+1}^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \boldsymbol{S}_{N+1}\boldsymbol{\phi}(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \left(\boldsymbol{S}_N^{-1} + \beta\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T\right)^{-1}\boldsymbol{\phi}(x) = \beta^{-1} +$$

$$\boldsymbol{\phi}(x)^T \left[\boldsymbol{S}_N - \frac{\beta\left(\boldsymbol{S}_N\boldsymbol{\phi}(x_n)\right)\left(\boldsymbol{\phi}(x_n)^T\boldsymbol{S}_N\right)}{1 + \beta\boldsymbol{\phi}(x_n)^T\boldsymbol{S}_N\boldsymbol{\phi}(x_n)}\right]\boldsymbol{\phi}(x) = \sigma_N^2(x) - \frac{\beta\left(\boldsymbol{\phi}(x_n)^T\boldsymbol{S}_N\boldsymbol{\phi}(x)\right)^2}{1 + \beta\boldsymbol{\phi}(x_n)^T\boldsymbol{S}_N\boldsymbol{\phi}(x_n)} \le \sigma_N^2(x)$$

Predictive Distribution: Summary

- The notation used here is as follows:

$$p(t \mid x, \mathbf{x}, \mathbf{t}) = \mathcal{N}(t \mid m(x), \sigma_N^2(x))$$

$$m(x) = \beta \boldsymbol{\phi}(x)^T \mathbf{S}_N \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) t_n$$

$$\sigma_N^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \mathbf{S}_N \boldsymbol{\phi}(x)$$

$$\mathbf{S}_N^{-1} = \alpha \mathbf{I} + \beta \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) \boldsymbol{\phi}(x_n)^T$$

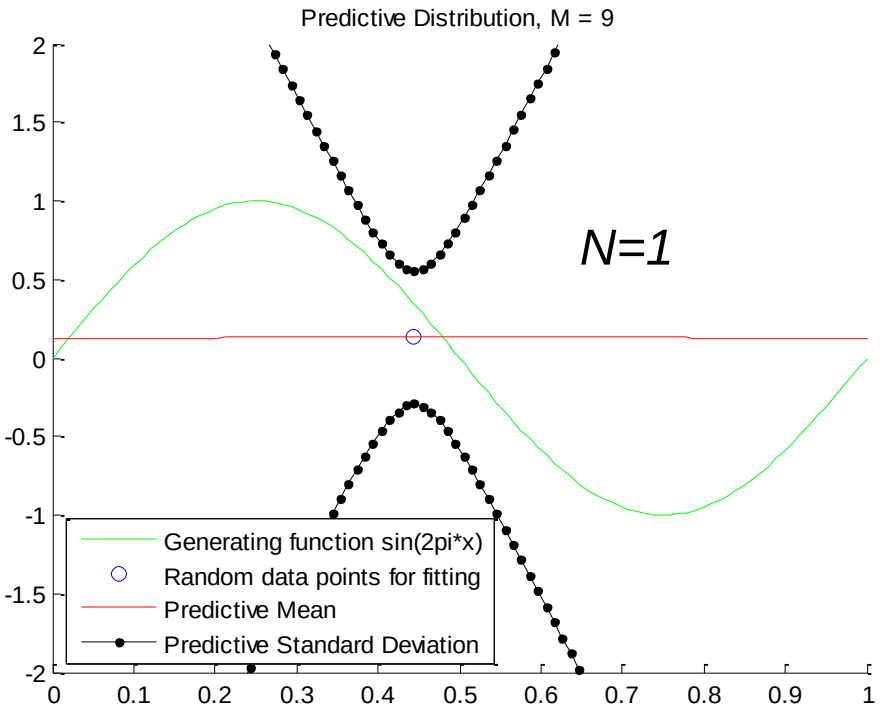
Note:
Predictive mean and variance are functions of x.

$$\boldsymbol{\phi}(x_n) = \begin{Bmatrix} \phi_0(x_n) \\ \phi_1(x_n) \\ \phi_2(x_n) \\ : \\ \phi_{M-1}(x_n) \end{Bmatrix}, \boldsymbol{\phi}(x)^T = \{\phi_0(x_n) \quad \phi_1(x_n) \quad \phi_2(x_n) \quad .. \quad \phi_{M-1}(x_n)\}, \mathbf{I} = \text{unit matrix } M \times M$$

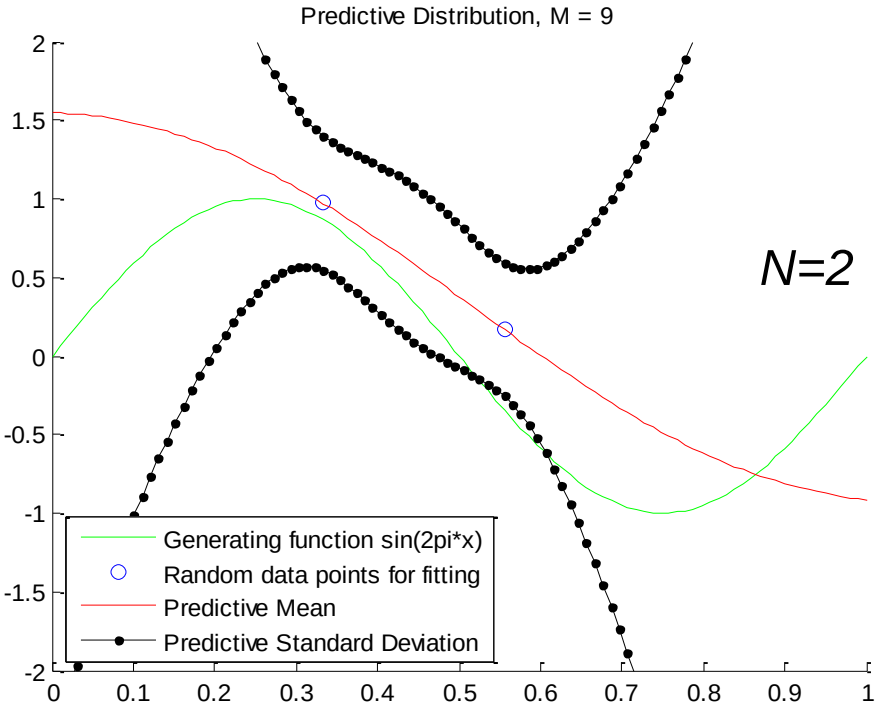
For Polynomial regression:

$$\boldsymbol{\phi}(x_n) = \begin{Bmatrix} 1 \\ x_n \\ x_n^2 \\ : \\ x_n^{M-1} \end{Bmatrix}, \boldsymbol{\phi}(x)^T = \{1 \quad x \quad x^2 \quad .. \quad x^{M-1}\}$$

Pointwise Uncertainty in the Predictions



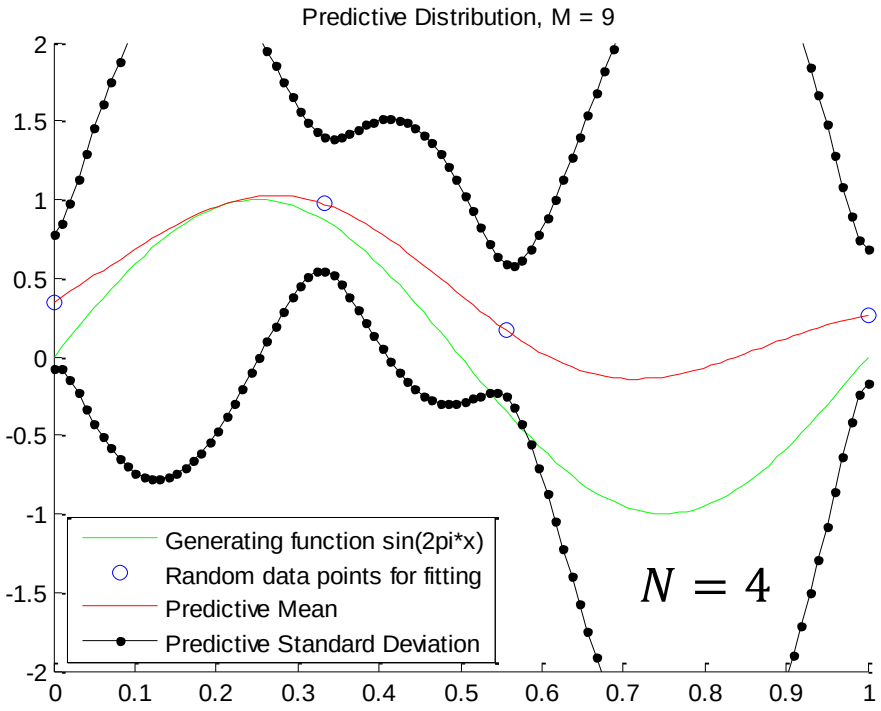
M=9 Gaussians, 10 parameters
Scale of Gaussians adjusted with data
$\alpha = 5 \times 10^{-3}$
$\beta = 11.1$
Using $N = 1,2,4,10$
Data are given here

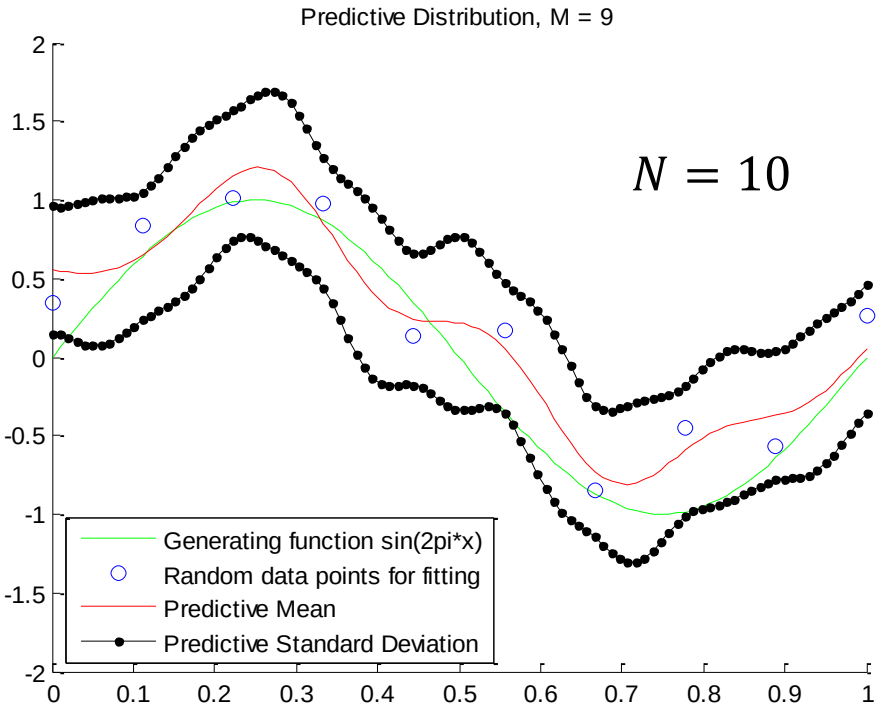


- The predictive uncertainty is smaller near the data.
- The level of uncertainty decreases with N

MatLab code

Pointwise Uncertainty in the Predictions

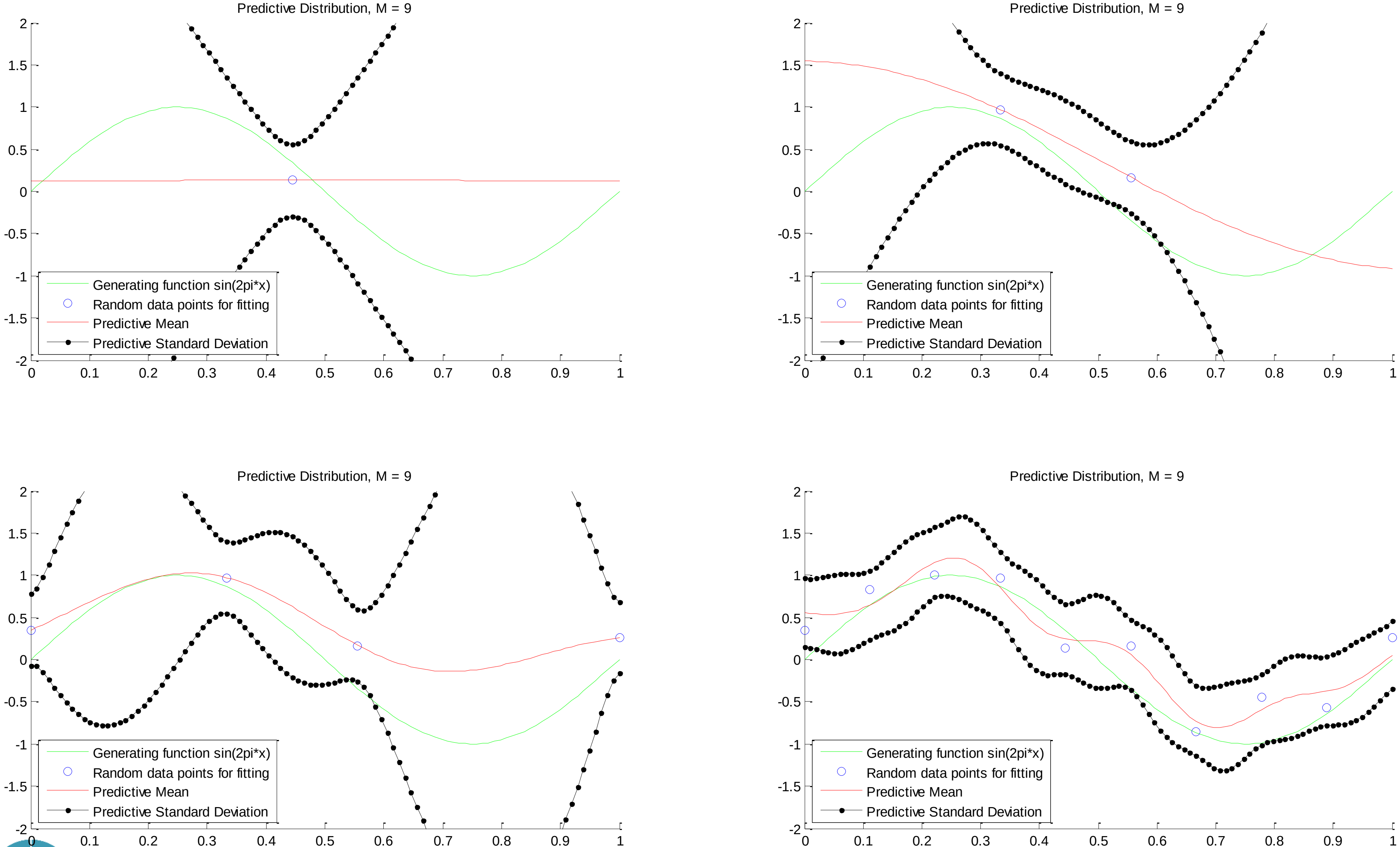




MatLab code

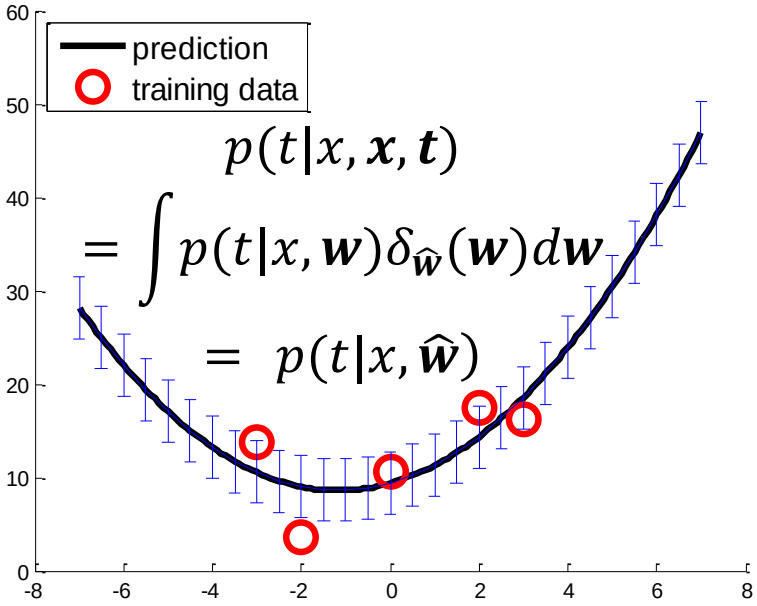
Summary of Results

MatLab code



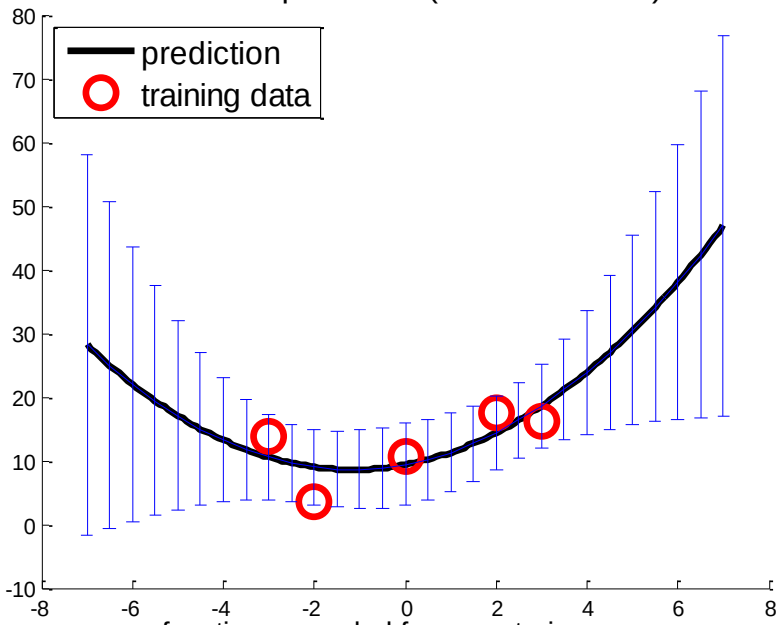
Plugin Approximation

plugin approximation (MLE)

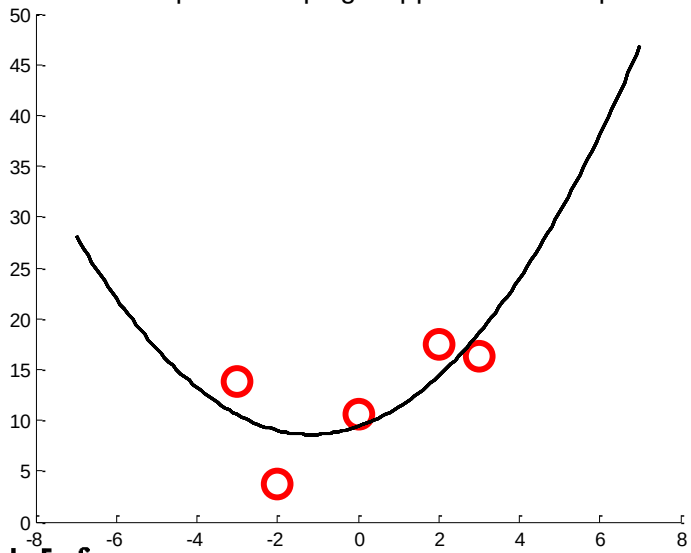


$$p(t|x, \boldsymbol{x}, \boldsymbol{t})$$
$$= \int p(t|x, \boldsymbol{w}) \delta_{\hat{\boldsymbol{w}}}(\boldsymbol{w}) d\boldsymbol{w}$$
$$= p(t|x, \hat{\boldsymbol{w}})$$

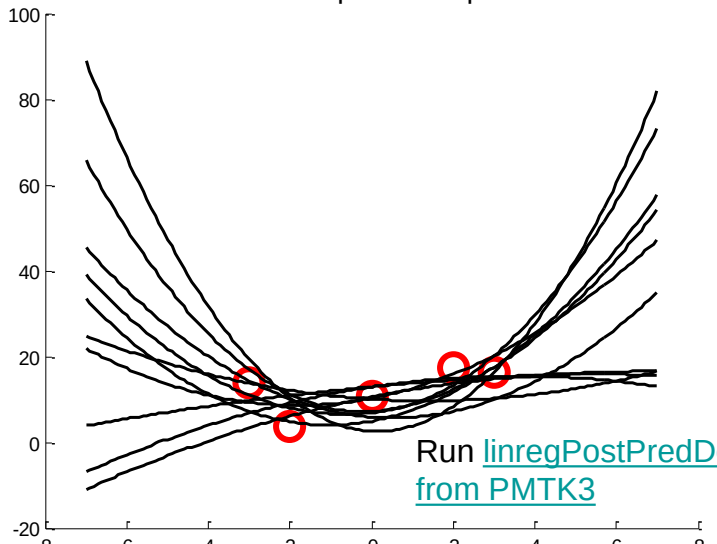
Posterior predictive (known variance)



functions sampled from plugin approximation to posterior



functions sampled from posterior



Run linregPostPredDemo from PMTK3

Predictive Distribution

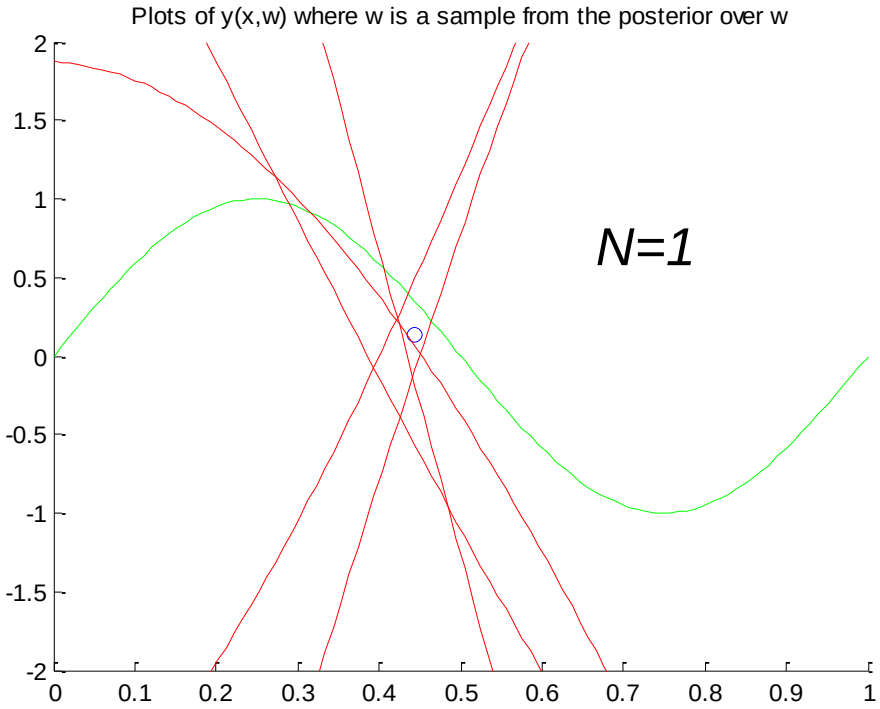
- We are not interested in $\boldsymbol{w}$ itself but in making predictions of t for new values of x. This requires the predictive distribution

$$p(t \mid x, \mathbf{x}, \mathbf{t}) = \int p(t \mid x, \boldsymbol{w}, \beta)\, p(\boldsymbol{w} \mid \mathbf{x}, \mathbf{t}, \alpha, \beta) d\boldsymbol{w} = \mathcal{N}\left(t \mid \boldsymbol{m}_N^T \boldsymbol{\phi}(x), \sigma_N^2(x)\right)$$

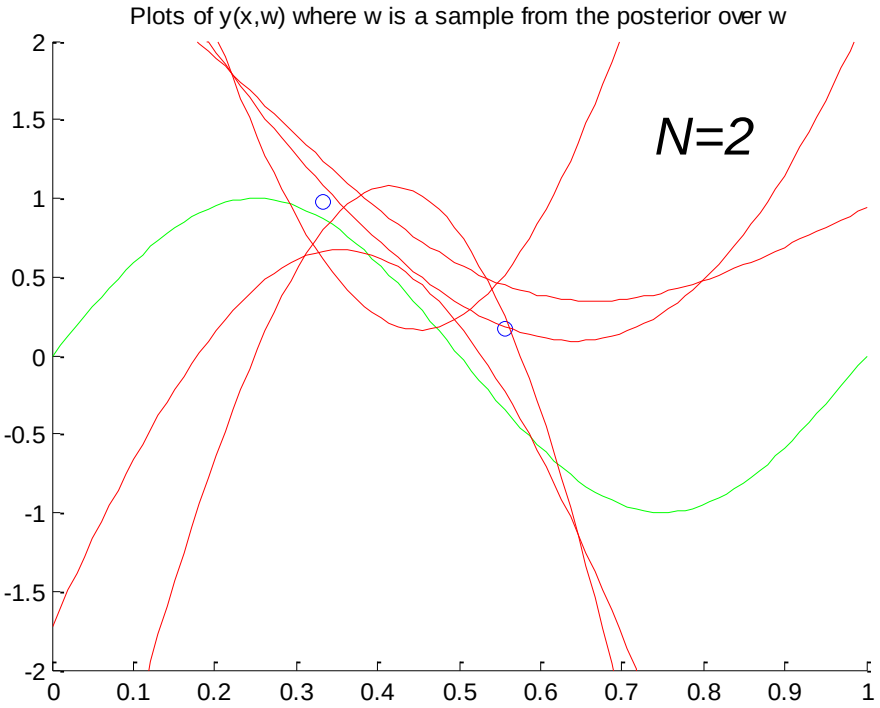
where $$\sigma_N^2(x) = \frac{1}{\beta} + \boldsymbol{\phi}(x)^T S_N \boldsymbol{\phi}(x), \quad \boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$$

- The 1st term represents the noise on the data whereas the 2nd term reflects the uncertainty associated with $\boldsymbol{w}$.
- Because the noise process and the distribution of $\boldsymbol{w}$ are independent Gaussians, their variances are additive.
- The error bars get larger as we move away from the training points. By contrast, in the plug-in approximation, the error bars are of constant size.
- As additional data points are observed, the posterior distribution becomes narrower.

Covariance Between the Predictions



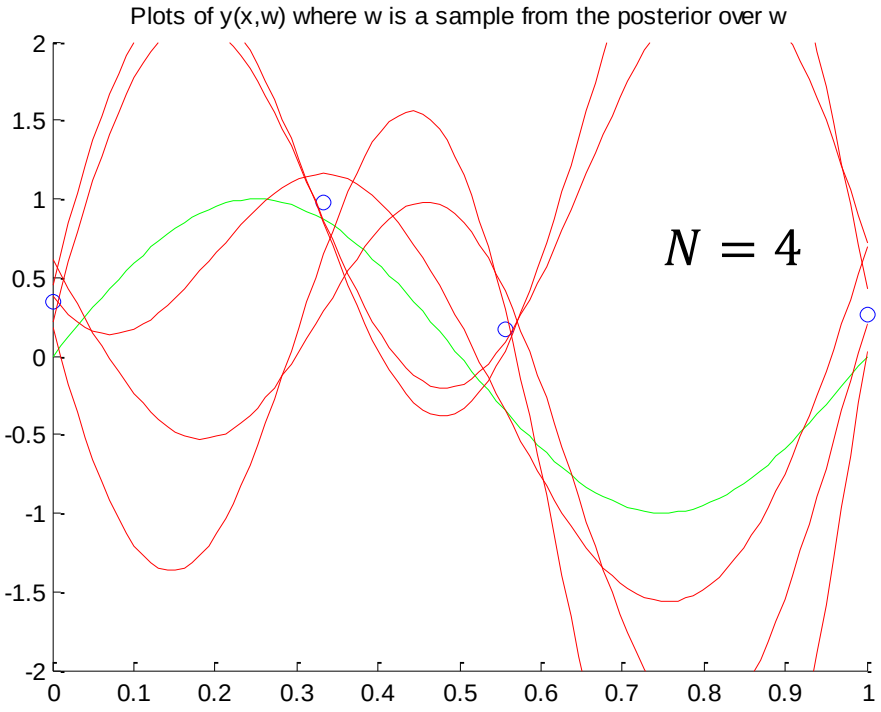
- Draw samples from the posterior of $\boldsymbol{w}$ and then plot $y(x, \boldsymbol{w})$. We use the same data as the earlier example.
- We are visualizing the joint uncertainty in the posterior distribution between the y values at two or more x values.



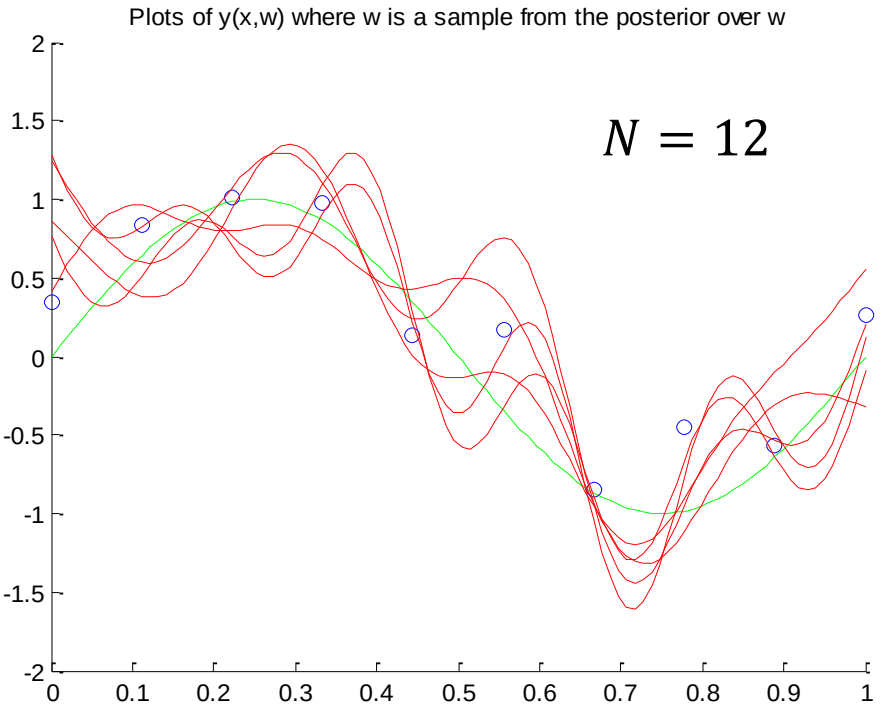
Same data and basis functions as in the earlier example

MatLab Code

Covariance Between the Predictions



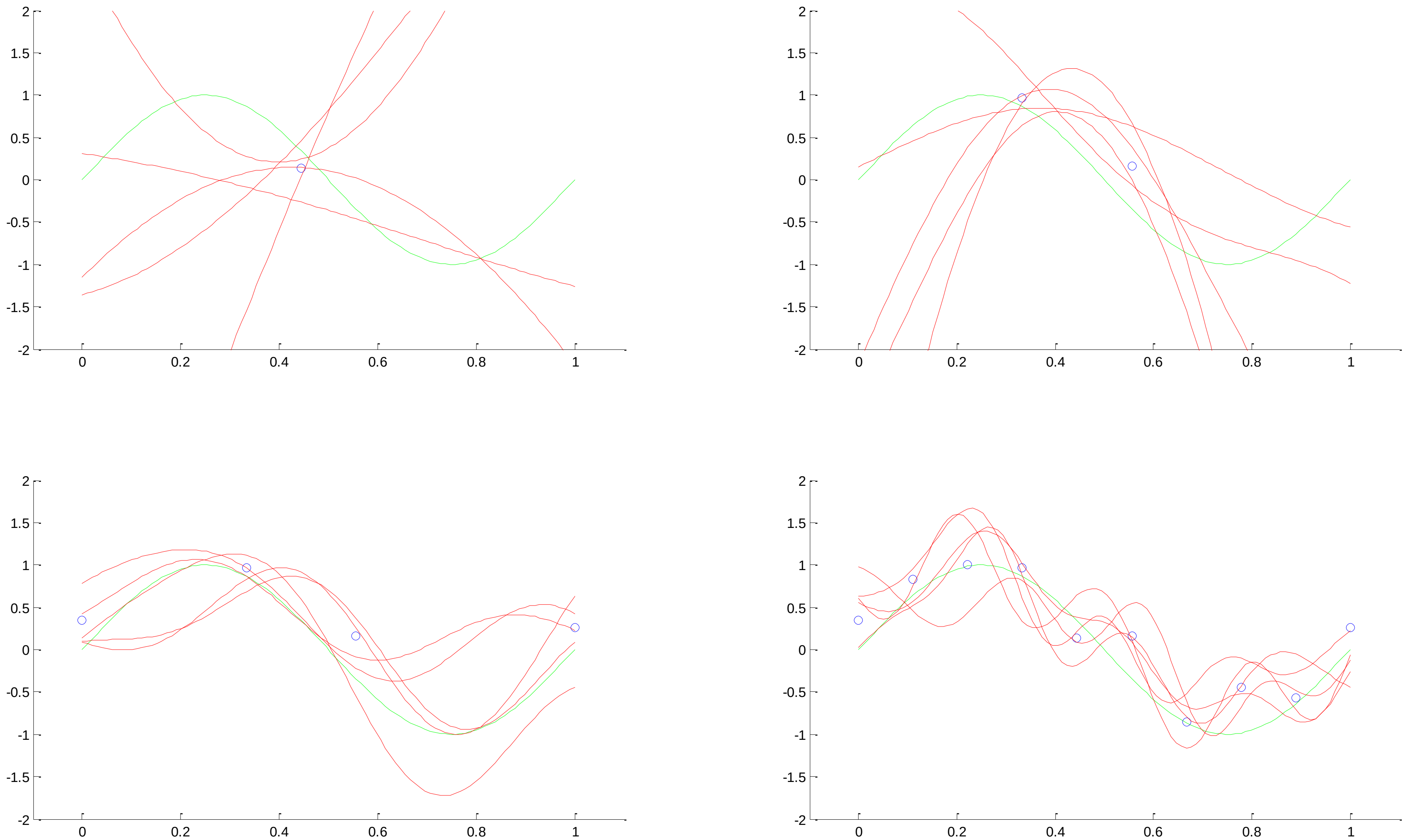
- Draw samples from the posterior of $\boldsymbol{w}$ and then plot $y(x, \boldsymbol{w})$
- We are visualizing the joint uncertainty in the posterior distribution between the y values at two or more x values.



MatLab Code

Summary of Results

MatLab Code



Gaussian Basis vs. Gaussian Process

- If we use localized basis functions such as Gaussians, then in regions away from the basis function support, the contribution from the second term in the predictive variance will go to zero, leaving only the noise contribution β^{-1}.

$$\sigma_N^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \boldsymbol{S}_N \boldsymbol{\phi}(x) \underset{\text{support of } \boldsymbol{\phi}(x)}{\overset{\text{away from the}}{\rightarrow}} \beta^{-1}$$

- The model becomes very confident in its predictions when extrapolating outside the region occupied by the basis functions. This is an undesirable behavior.

- This problem can be avoided by adopting an alternative Bayesian approach to regression (Gaussian processes).

Equivalent Kernel

- The predictive mean in our regression model can be written as:

$$y(\boldsymbol{x}, \boldsymbol{m}_N) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{m}_N = \beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t} = \sum_{n=1}^{N} \underbrace{\beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}_n)}_{\text{Equivalent kernel } k(\boldsymbol{x}, \boldsymbol{x}_n)} t_n \Rightarrow$$

$$y(\boldsymbol{x}, \boldsymbol{m}_N) = \sum_{n=1}^{N} k(\boldsymbol{x}, \boldsymbol{x}_n) t_n$$

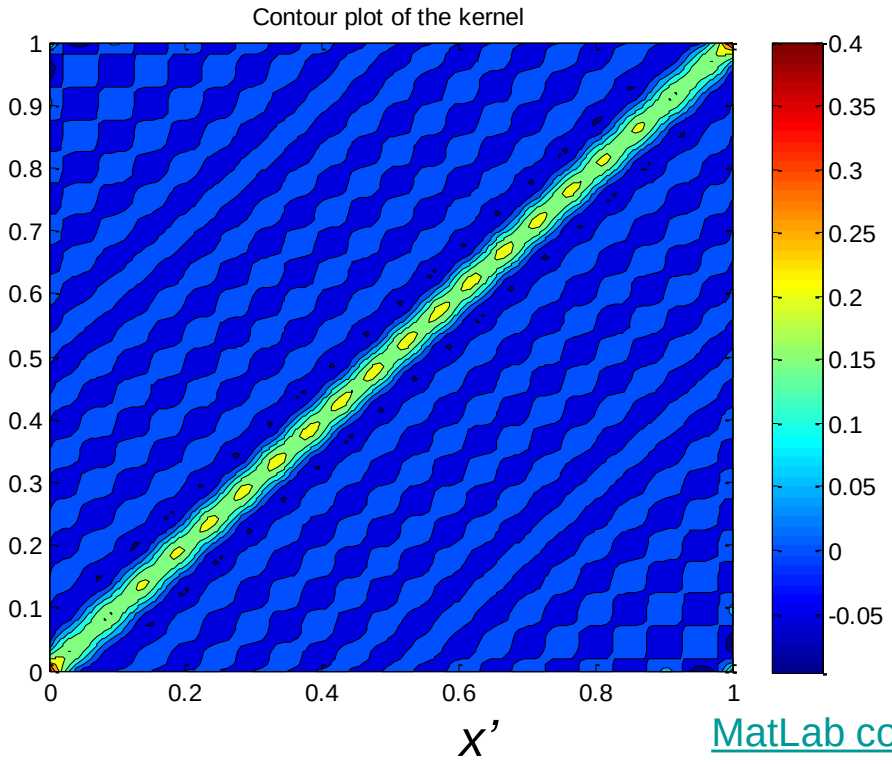
$$k(\boldsymbol{x}, \boldsymbol{x}_n) = \beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}_n)$$

The kernel is shown here as a plot of $\boldsymbol{x}$ *vs.* $\boldsymbol{x}'$.

20 *samples* x_n equally spaced in [0,1]

Gaussian kernels ($s = 0.05$), $\alpha = 5 \times 10^{-3}$, $\beta = 11.1$.

$$\phi_j(x) = \exp\left(-\frac{(x - x_j)^2}{2s^2}\right)$$



MatLab code

Equivalent Kernel

- The predictive mean can be written as follows:

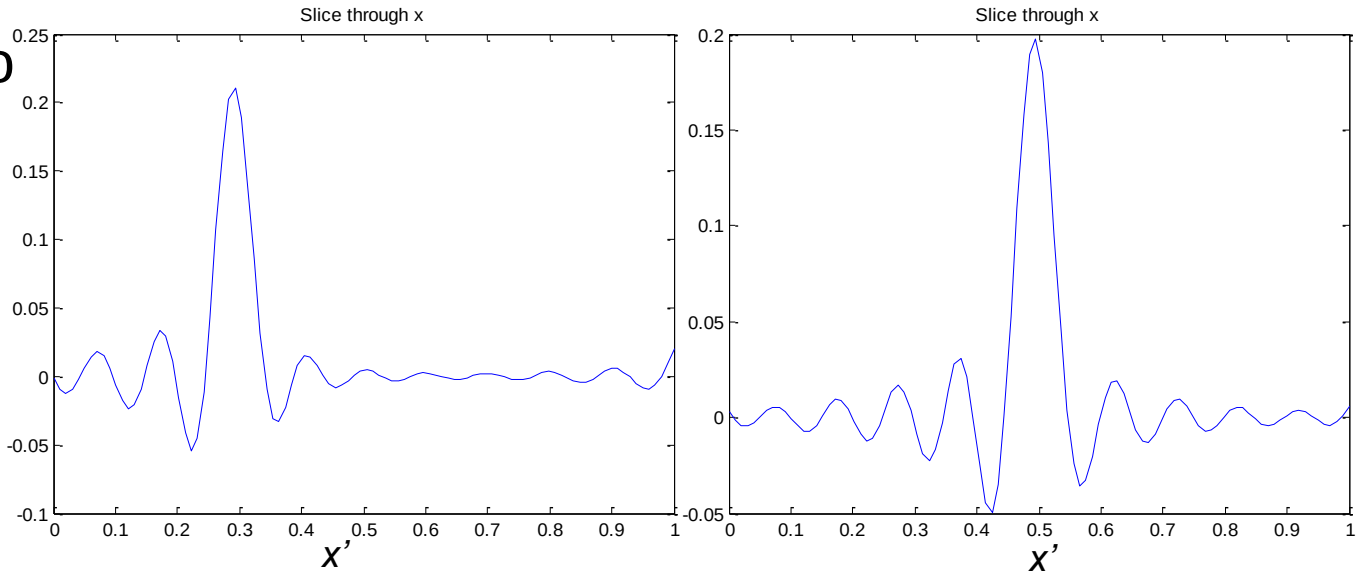
$$y(\boldsymbol{x}, \boldsymbol{m}_N) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{m}_N = \beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t} = \sum_{n=1}^{N} \underbrace{\beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}_n)}_{\text{Equivalent kernel } k(\boldsymbol{x}, \boldsymbol{x}_n)} t_n \Rightarrow$$

$$y(\boldsymbol{x}, \boldsymbol{m}_N) = \sum_{n=1}^{N} k(\boldsymbol{x}, \boldsymbol{x}_n) t_n \qquad k(\boldsymbol{x}, \boldsymbol{x}_n) = \beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}_n)$$

Two slices corresponding to $x = 0.3, x = 0.5$.

20 Gaussians ($s = 0.05$)

Plots for 100 points x uniformly in (0,1)

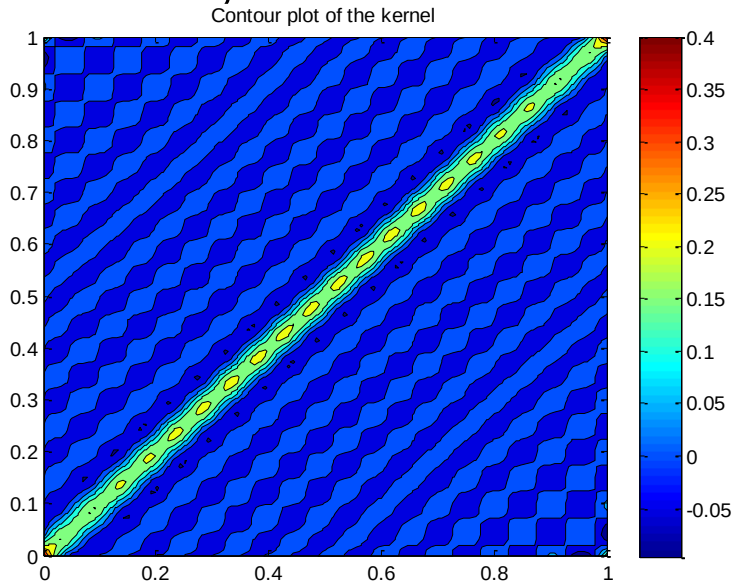


MatLab code

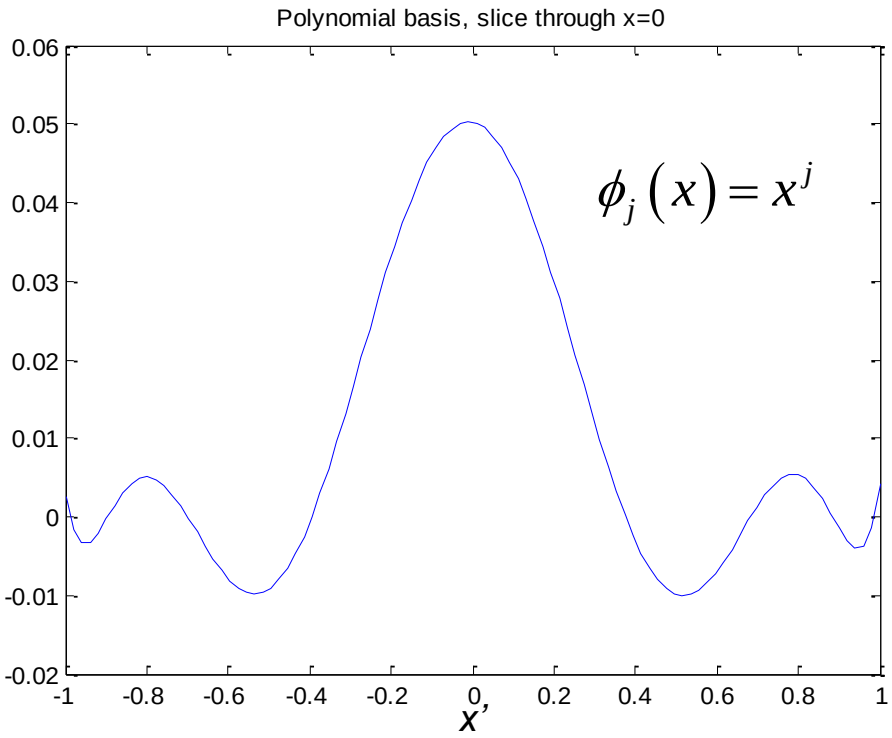
Equivalent Kernel

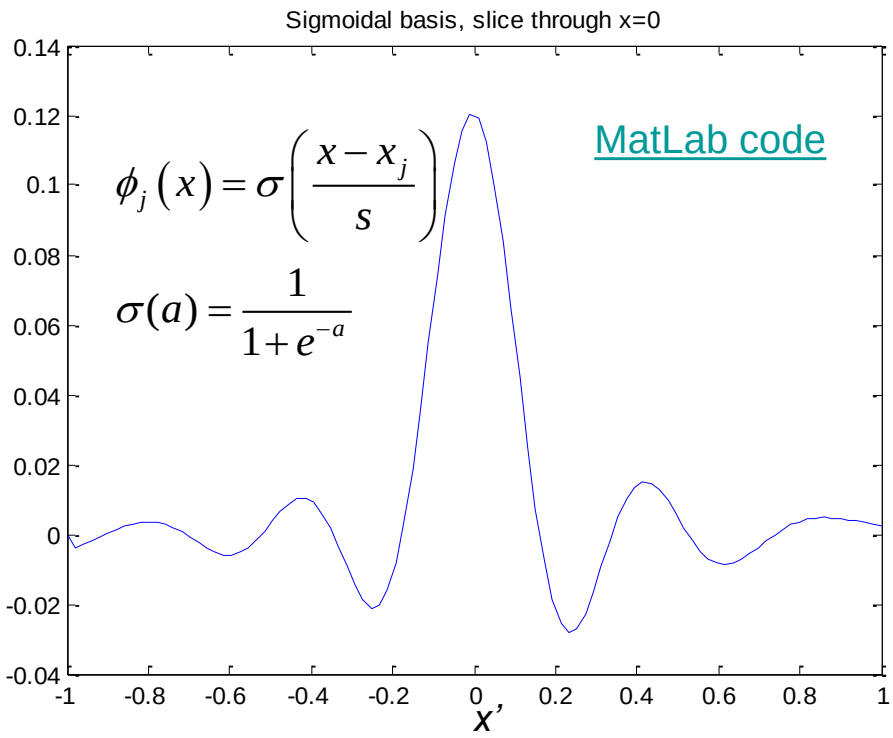
$$y(\boldsymbol{x}, \boldsymbol{m}_N) = \sum_{n=1}^{N} k(\boldsymbol{x}, \boldsymbol{x}_n) t_n \qquad k(\boldsymbol{x}, \boldsymbol{x}_n) = \beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}_n)$$

- Note that we make predictions by taking linear combinations of the training set target values (*linear smoothers*).
- The equivalent kernel depends on the input values $\boldsymbol{x}_n$ since these appear in $\boldsymbol{S}_N$.
- See from the figure that its localized around x, and so the mean of the predictive distribution $y(\boldsymbol{x}, \boldsymbol{m}_N)$, is obtained by forming a weighted combination of t_n in which data points close to $\boldsymbol{x}$ are given higher weight than points (evidence) further removed from $\boldsymbol{x}$.



Equivalent Kernel





- Examples of equivalent kernels $k(x, x')$ for $x = 0$ plotted as a function of x'. 10 basis functions were used in each case. The sigmoidal basis is centered at 10 equally spaced x_n in $[-1,1]$.
- These are localized functions of x' even though the corresponding basis functions are nonlocal.

Equivalent Kernel

- Consider the covariance between $y(\boldsymbol{x})$ and $y(\boldsymbol{x}')$, which is given by (recall that $p(\boldsymbol{w}\mid \boldsymbol{x},\boldsymbol{t},\alpha,\beta)=\mathcal{N}\left(\boldsymbol{w}\mid,\beta\boldsymbol{S}_N\sum_{n=1}^{N}t_n\boldsymbol{\phi}(x_n),\boldsymbol{S}_N\right)$):

$$\operatorname{cov}\left[y(\boldsymbol{x}),y(\boldsymbol{x}')\right]=\operatorname{cov}\left[\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{w},\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}')\right]=\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}')=\beta^{-1}k(\boldsymbol{x},\boldsymbol{x}')$$

- From the earlier discussion on the equivalent kernel, we see that the predictive mean at nearby points will be highly correlated, whereas for more distant points the correlation will be smaller.
- By drawing samples from the posterior distribution over $\boldsymbol{w}$, and plotting the corresponding model functions $y(\boldsymbol{x},\boldsymbol{w})$, we are visualizing the joint uncertainty in the posterior distribution between the y values at two (or more) $\boldsymbol{x}$ values, as governed by the equivalent kernel.
- This is contrary to the Figure here where we visualized pointwise uncertainty in the predictions.

Equivalent Kernel

- We can avoid the use of basis functions and define the kernel function directly, leading e.g. to *Gaussian Processes* (more on Kernel methods later on in this course).
- *Note that*

$$\sum_{n=1}^{N} k(\boldsymbol{x}, \boldsymbol{x}_n) = 1$$

for all values of $\boldsymbol{x}$. However, the equivalent kernel may be negative for some values of $\boldsymbol{x}$.

- Like all kernel functions, the equivalent kernel can be expressed as an inner product:

$$k(\boldsymbol{x}, \boldsymbol{z}) = \beta \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{z}) \Rightarrow \quad k(\boldsymbol{x}, \boldsymbol{z}) = \psi(\boldsymbol{x})^T \psi(\boldsymbol{z})$$
$$\psi(\boldsymbol{x}) = \beta^{1/2} \boldsymbol{S}_N^{1/2} \boldsymbol{\phi}(\boldsymbol{x})$$

Computing the Bias Parameter

- If we make the bias parameter w_0 explicit, then the error function becomes

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - w_0 - \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x}_n)\right)^2$$

- Setting the derivative with respect to w_0 equal to zero, and solving for w_0, we obtain

$$w_0 = \frac{1}{N}\sum_{n=1}^{N} t_n - \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x}_n) \Rightarrow$$

$$w_0 = \bar{t} - \sum_{j=1}^{M} w_j \bar{\phi}_j, \quad \bar{t} = \frac{1}{N}\sum_{n=1}^{N} t_n, \quad \bar{\phi}_j = \frac{1}{N}\sum_{n=1}^{N} \phi_j(\boldsymbol{x}_n)$$

- The bias parameter w_0 compensates for the difference between the averages of the target values and the weighted sum of the averages of the basis function values.

A Note on Data Centering

- In linear regression, it helps to center the data in a way that does not require us to compute the offset term $w_0 \equiv \mu$. Write the likelihood as:

$$p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{w}, \mu, \beta) \propto \exp\left(-\frac{\beta}{2} \left(\boldsymbol{t} - \mu \mathbf{1}_N - \boldsymbol{\Phi} \boldsymbol{w} \right)^T \left(\boldsymbol{t} - \mu \mathbf{1}_N - \boldsymbol{\Phi} \boldsymbol{w} \right) \right)$$

- Let us assume that the input data are centered in each dimension such that:

$$\sum_{j=1}^{N} \phi_i \left(\boldsymbol{x}_j \right) = 0 \; \forall \; i = 1, \ldots, M-1$$

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}^T(\boldsymbol{x}_1) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ : \\ \boldsymbol{\phi}^T(\boldsymbol{x}_N) \end{pmatrix} = \begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \phi_2(\boldsymbol{x}_1) & .. & \phi_{M-1}(\boldsymbol{x}_1) \\ \phi_1(\boldsymbol{x}_2) & \phi_2(\boldsymbol{x}_2) & .. & \phi_{M-1}(\boldsymbol{x}_2) \\ : & : & : & : \\ \phi_1(\boldsymbol{x}_N) & \phi_2(\boldsymbol{x}_N) & .. & \phi_{M-1}(\boldsymbol{x}_N) \end{pmatrix},$$

$$\boldsymbol{\Phi}^T = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_2) & .. & \boldsymbol{\phi}(\boldsymbol{x}_N) \end{pmatrix}$$

$$\boldsymbol{\phi}(\boldsymbol{x}_i) \equiv \begin{pmatrix} \phi_1(\boldsymbol{x}_i) & \phi_2(\boldsymbol{x}_i) & . & . & \phi_{M-1}(\boldsymbol{x}_i) \end{pmatrix}^T$$

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\varphi}_1 & \boldsymbol{\varphi}_2 & .. & \boldsymbol{\varphi}_{M-1} \end{pmatrix}$$

$$\boldsymbol{\varphi}_i \equiv \begin{pmatrix} \phi_i(\boldsymbol{x}_1) & \phi_i(\boldsymbol{x}_2) & . & . & \phi_i(\boldsymbol{x}_N) \end{pmatrix}^T$$

- The mean of the output is equally likely to be positive or negative. Let us put an improper prior $p(\mu) \propto 1$ and integrate μ out.

A Note on Data Centering

- Introducing, $\bar{t} = \frac{1}{N}\sum_{i=1}^{N} t_i$, the marginal likelihood becomes :

$$p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{w}, \beta) \propto \int \exp\left(-\frac{\beta}{2} \left(\underbrace{\boldsymbol{t} - \bar{t}\mathbf{1}_N - \boldsymbol{\Phi}\boldsymbol{w}}_{\boldsymbol{A}} - \left(\mu - \bar{t}\right)\mathbf{1}_N \right)^T \left(\boldsymbol{t} - \bar{t}\mathbf{1}_N - \boldsymbol{\Phi}\boldsymbol{w} - \left(\mu - \bar{t}\right)\mathbf{1}_N \right) \right) d\mu$$

- Completing the square in μ gives (use the centering of $\boldsymbol{\Phi}$):

$$p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{w}, \beta) \propto \int \exp\left(-\frac{\beta}{2} \left(\left(\mu - \bar{t}\right)^2 N - 2\left(\mu - \bar{t}\right) \underbrace{\boldsymbol{A}^T \mathbf{1}_N}_{\bar{t}N - \bar{t}N - \boldsymbol{w}^T \underbrace{\boldsymbol{\Phi}^T \mathbf{1}_N}_{=0}} + \boldsymbol{A}^T \boldsymbol{A} \right) \right) d\mu$$

$$\propto \exp\left(-\frac{\beta}{2} \left(\boldsymbol{t} - \bar{t}\mathbf{1}_N - \boldsymbol{\Phi}\boldsymbol{w} \right)^T \left(\boldsymbol{t} - \bar{t}\mathbf{1}_N - \boldsymbol{\Phi}\boldsymbol{w} \right) \right)$$

- Our model is now simplified if instead of $\boldsymbol{t}$ we use (centered output) $\boldsymbol{t} \leftarrow \boldsymbol{t} - \bar{t}\mathbf{1}_N$ and the likelihood is simply written as:

$$p(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{w}, \beta) \propto \exp\left(-\frac{\beta}{2} \left(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w} \right)^T \left(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w} \right) \right)$$

- Recall that the MLE estimate for μ is: $\hat{\mu} = \bar{t} - \sum_{j=1}^{M} \bar{\boldsymbol{\phi}}_j^T \boldsymbol{w}_j$, $[\bar{\boldsymbol{\phi}}_1, \ldots, \bar{\boldsymbol{\phi}}_{M-1}]$ *is formed by taking the average of each column of* $\boldsymbol{\Phi}$

A Note on Data Centering

- As an example, consider a linear regression model of the form

$$\mathbb{E}[y \mid \boldsymbol{x}] = w_0 + \boldsymbol{w}^T \boldsymbol{x}$$

- In the context e.g. of MLE, we need to minimize

$$\min_{w_0, \boldsymbol{w}} = \sum_{i=1}^{N} \left(t_i - w_0 - \boldsymbol{w}^T \boldsymbol{x}_i \right)^2$$

- Minimization wrt w_0 gives:

$$\sum_{i=1}^{N} \left(t_i - w_0 - \boldsymbol{w}^T \boldsymbol{x}_i \right) = 0 \Rightarrow w_0 N = \bar{t} N - N \boldsymbol{w}^T \bar{\boldsymbol{x}} \Rightarrow w_0 = \bar{t} - \boldsymbol{w}^T \bar{\boldsymbol{x}}$$

where:

$$\bar{\boldsymbol{x}} \equiv \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_M \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} x_{i1} \Big/ N \\ \sum_{i=1}^{N} x_{i2} \Big/ N \\ \vdots \\ \sum_{i=1}^{N} x_{iM} \Big/ N \end{pmatrix}, \quad \bar{t} \equiv \sum_{i=1}^{N} t_i \Big/ N$$

- Thus:

$$\hat{w}_0 = \bar{t} - \bar{\boldsymbol{x}}^T \boldsymbol{w}$$

A Note on Data Centering

- Substituting the bias term in our objective function gives:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{N} \left(t_i - \bar{t} + \boldsymbol{w}^T \bar{\boldsymbol{x}} - \boldsymbol{w}^T \boldsymbol{x}_i \right)^2 = \min_{\boldsymbol{w}} \sum_{i=1}^{N} \left(t_i - \bar{t} + \boldsymbol{w}^T \left(\bar{\boldsymbol{x}} - \boldsymbol{x}_i \right) \right)^2$$

- Minimization wrt $\boldsymbol{w}$ gives:

$$\sum_{i=1}^{N} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \hat{\boldsymbol{w}} = \sum_{i=1}^{N} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{t}_i - \bar{\boldsymbol{t}})$$

- We thus first compute the MLE of $\boldsymbol{w}$ using the centered input and output as follows:

$$\hat{\boldsymbol{w}} = \left(\boldsymbol{X}_c^T \boldsymbol{X}_c \right)^{-1} \boldsymbol{X}_c^T \boldsymbol{t}_c = \left[\sum_{i=1}^{N} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \right] \left[\sum_{i=1}^{N} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{t}_i - \bar{\boldsymbol{t}}) \right]$$

$$\boldsymbol{X}_c = \boldsymbol{X} - \bar{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{1}_N \bar{\boldsymbol{x}}^T = \begin{pmatrix} \boldsymbol{x}_1^T - \bar{\boldsymbol{x}}^T \\ \boldsymbol{x}_2^T - \bar{\boldsymbol{x}}^T \\ \vdots \\ \boldsymbol{x}_N^T - \bar{\boldsymbol{x}}^T \end{pmatrix} = \begin{pmatrix} x_{11} - \bar{x}_1 & x_{12} - \bar{x}_2 & .. & x_{1M} - \bar{x}_M \\ x_{21} - \bar{x}_1 & x_{22} - \bar{x}_2 & .. & x_{2M} - \bar{x}_M \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} - \bar{x}_1 & x_{N2} - \bar{x}_2 & .. & x_{NM} - \bar{x}_M \end{pmatrix}, \quad \bar{\boldsymbol{x}} \equiv \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_M \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} x_{i1} \Big/ N \\ \sum_{i=1}^{N} x_{i2} \Big/ N \\ \vdots \\ \sum_{i=1}^{N} x_{iM} \Big/ N \end{pmatrix}, \quad \boldsymbol{t}_c = \boldsymbol{t} - \bar{\boldsymbol{t}} = \boldsymbol{t} - \boldsymbol{1}_N \bar{t}, \quad \bar{t} \equiv \sum_{i=1}^{N} t_i \Big/ N$$

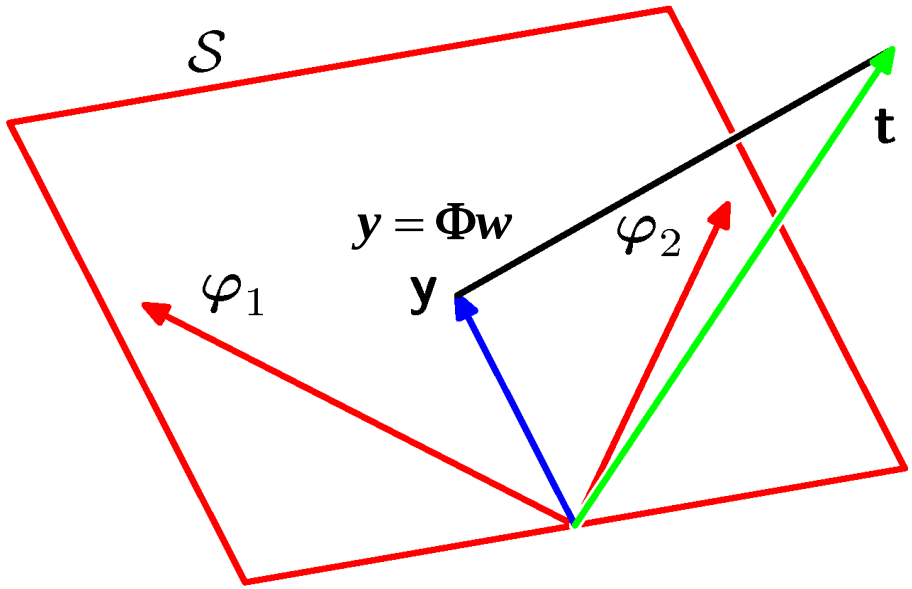
- We can then estimate the MLE estimate of w_0 as follows:

$$\hat{w}_0 = \bar{t} - \bar{\boldsymbol{x}}^T \boldsymbol{w}$$

Geometry of Least Squares

- We look for a geometrical interpretation of the least-squares solution in an N-dimensional space. $\boldsymbol{t}$ is a vector in that space with components $t_1, \ldots, t_N$ $(N > M)$.
- The least-squares regression function is obtained by finding the orthogonal projection of the data vector $\boldsymbol{t}$ onto the subspace spanned by the basis functions $\boldsymbol{\varphi}_j(\boldsymbol{x})$.
- Note $\boldsymbol{\varphi}_j$ is here the j –th column of $\boldsymbol{\Phi}$.

$$\boldsymbol{\varphi}_j \equiv \left(\phi_j(\boldsymbol{x}_1), \ldots, \phi_j(\boldsymbol{x}_N)\right)^T$$



$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\varphi}_0 & \boldsymbol{\varphi}_1 & \cdots & \boldsymbol{\varphi}_{M-1} \end{pmatrix} = \begin{pmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & .. & \phi_{M-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & .. & \phi_{M-1}(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & .. & \phi_{M-1}(\boldsymbol{x}_N) \end{pmatrix}$$

Geometry of Least Squares

- We are looking for $\boldsymbol{w}$ such that the projection error

$$\boldsymbol{t} - \boldsymbol{y} = \boldsymbol{t} - \boldsymbol{\Phi w}$$

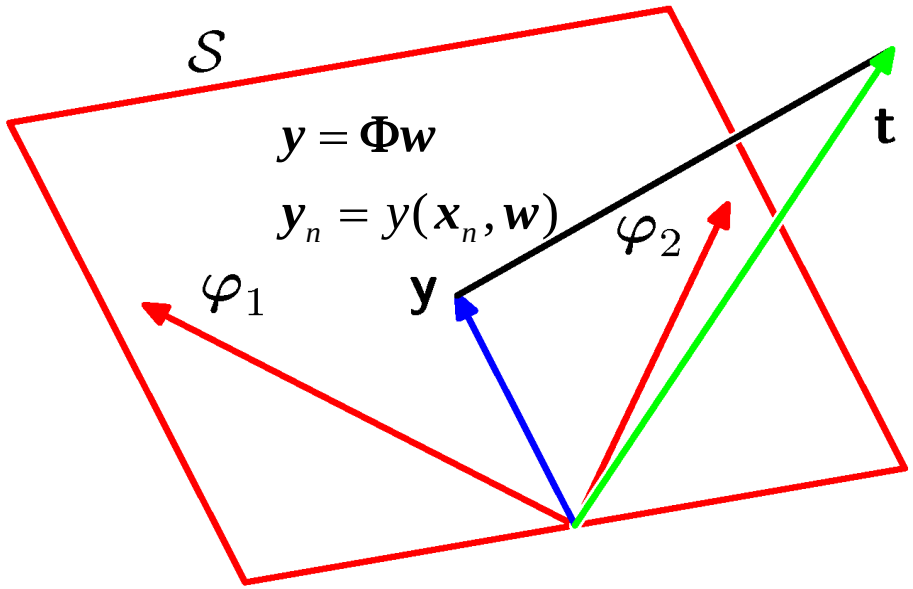
is orthogonal to the basis $\boldsymbol{\varphi}_j$, i.e. such that:

$$\boldsymbol{\Phi}^T \left(\boldsymbol{t} - \boldsymbol{\Phi w} \right) = 0$$

- These are the normal equations we derived earlier.

$$\boldsymbol{\Phi}^T = \begin{pmatrix} \boldsymbol{\varphi}_0^T \\ \boldsymbol{\varphi}_1^T \\ : \\ \boldsymbol{\varphi}_{M-1}^T \end{pmatrix}$$

$$\boldsymbol{\varphi}_j(\boldsymbol{x}) = \left(\phi_j(\boldsymbol{x}_1), ..., \phi_j(\boldsymbol{x}_N) \right)^T$$



$\mathcal{S}$: M –dimensional subspace spanned by $\boldsymbol{\varphi}_j(\boldsymbol{x})$

$$\boldsymbol{\Phi} = \left[\boldsymbol{\varphi}_0(\boldsymbol{x}) \quad \boldsymbol{\varphi}_1(\boldsymbol{x}) \quad .. \quad \boldsymbol{\varphi}_{M-1}(\boldsymbol{x}) \right]$$